Department of Computer Science

Technical Report

Undergraduate Dissertation: Opinion Sharing on the Web Uings a Distributed Architecture and Trusted Social Networks

Chris Beach

Technical Report 2005-06 **July 2005**

ISSN 1740-9497

Contact Address:
Department of Computer Science
University of Bath
Bath, BA2 7AY
United Kingdom
URL: http://www.cs.bath.ac.uk

ISSN 1740-9497

OPINION SHARING ON THE WEB USING A DISTRIBUTED ARCHITECTURE AND TRUSTED SOCIAL NETWORKS

CHRIS BEACH

BSC IN COMPUTER SCIENCE

MAY 12TH 2005

OPINION SHARING ON THE WEB USING A DISTRIBUTED ARCHITECTURE AND TRUSTED SOCIAL NETWORKS

SUBMITTED BY CHRIS BEACH

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed.....

This dissertation may be made available for consultation within the University Library and may be photocopied or lent to other libraries for the purposes of consultation.

Signed.....

Abstract

Online shops and service providers often require a leap of faith from the consumer. It is important to identify unreliable vendors. Whilst opinion-pooling sites offer a measure of reputation, they are subject to untraceable, anonymous votes. They also require several steps of interaction from the user.

A system is implemented to share reputation ratings between individuals using trusted social networks. A client, integrated into the user's web browser, shows the rating of a vendor website as it is browsed. The system distributes storage and computation load, making it inherently scalable. Additionally, the reputation data is exposed through Web Services, allowing interaction with other software clients and agent-based systems.

Current research in the field of recommender systems is considered, and issues of security, trust and privacy are investigated in the context of the implemented system.

Acknowledgements

I would like to thank my supervisor Julian Padget for his helpful insight and direction into the research surrounding the project

Contents

1 Introduction

After the `dot com' explosion of the 90's we find ourselves in a world where almost anything can be purchased online. Many online retailers have made a good name for themselves. However, blind faith is required when making purchases from less well-known sources, particularly if they are not based in this country (and not subject to stringent e-commerce law).

Dooyoo[1], Dealtime[2], and various opinion-polling web sites offer feedback to help the buyer form an opinion. However, there are a number of issues with such sites. Their independence is not guaranteed, particularly as many of them are subject to commercial sponsorship. It is possible to place multiple ratings against the same vendor, and there is no mechanism to prevent an individual promoting their own site. From the user's point of view, several steps are required to use opinion-polling sites to discover the reputation of a vendor. Likewise, before submitting ratings, a user generally is required to fill forms and create an account. Can this lack of trust, immediacy and usability be addressed?

This paper describes the implementation of a system that provides an instant and highly trustworthy means of finding the reputation of an online retailer. The design, technical implementation and testing will be described in detail. The final chapters will critically evaluate the system and draw conclusions.

In terms of development style, the evolutionary methodology has been used throughout the project. This paradigm offers many advantages over traditional approaches such as the waterfall model, which impose rigid structures and fixed deliverables. Instead, throw-away prototyping, rapid development and a non-linear development of ideas have all been key to the success of this project.

[1] http://www.dooyoo.com

[2] http://www.dealtime.com

2 Literature Survey

This chapter will survey the current state of research in fields related to the proposed recommender system. The survey will focus on areas where existing research can be expanded and applied, and acknowledge the original sources.

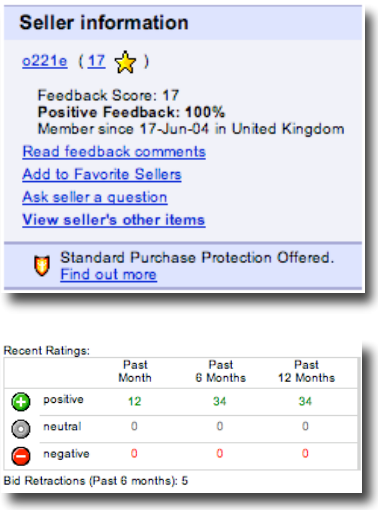
The proposed system draws from the following fields:

- Measurement of trust/reputation
- Distributed architecture
- Social networks
- The application of semantics to the web

Reputation Mechanisms in eCommerce

eBay

The world's largest online auction, eBay[3], has a well-established mechanism for recording the reputation of sellers. After a transaction has been completed, the buyer and seller are invited to leave a rating and comment for each other indicating their satisfaction with the transaction. These ratings and comments are then publicly available to other eBay users, who can use it to decide whether to deal with a particular individual.



At least two independent studies have shown that the reputation ratings have an effect on the success of eBay sellers. In a controlled experiment, Resnick et al demonstrate that buyers were willing to pay a strong-reputation seller 8.1% more on average than a new seller for a given item [10]. Likewise, Melnik and Alm have shown a statistical significance between reputation ratings and auction end price [7]. However, both studies show only a moderate correlation between seller success and reputation, considering the extremes in reputation tested. This may be a result of several factors, including eBay's various buyer protection mechanisms that negate some of the risk involved in transactions with low-reputation users. It is likely that being associated with a well-known brand such as eBay will give all users a baseline level of perceived reputation.

On the Internet at large, no such baseline exists. In the absence of word-of-mouth recommendation, a user may judge the trustworthiness of a web site merely by superficial factors such as its visual appearance. [3]

[3] http://www.ebay.com

Epinions

Epinions[4] is an online review site for products and services. It is innovative in allowing reviewers to be themselves reviewed. This mechanism allows users to become trusted peers, and gain authority for their opinions. Whilst Epinions is a promising system, it relies on users performing several time-consuming steps (search, browse ratings, consider bias) before getting a rating. Also, ratings from anonymous users with potential vested interests are still considered.

Trust Mechanisms applied to Email Filtering

With spam becoming widespread and problematic, a great deal of investment has been made in systems that can catch and prevent it. Golbeck et al propose an email client, TrustMail, that uses knowledge of social networks to filter incoming mail. [4] This is similar to the "whitelist" mechanism, whereby a finite list of known email senders is built, and mail from an unknown source is automatically moved to a low priority list or automatically deleted. Users may assign a level of trust to known contacts, and using a recursive algorithm, the system will aggregate trust ratings for contacts on other people's lists.

Many common email clients support filtering by whitelist, but TrustMail automates this process by querying existing social networks to build the whitelist. There is, however, a burden on the user in adding trust ratings for the contacts. This is reduced by the inferred ratings that propagate through the social network.

For TrustMail the social network is constructed from FOAF data, extended to allow the storing of reputation data against people. However, as discussed earlier, the adoption of Semantic Web technologies is largely restricted to the academic community and FOAF is not ubiquitous on the World Wide Web (WWW). To be useful on the WWW, Golbeck et al. suggest that existing social networks such as Microsoft Networks (MSN), Yahoo! Mail, and America Online (AOL) might be adapted to allow additional meta-data to be stored on users.

Unfortunately the authors of the paper appear not have implemented a TrustMail client and therefore can draw only limited conclusions on its possible effectiveness.

Trust and Reputation in Agent-Based Societies

Multi-Agent Systems (MAS) is a software field empowered by the rise of ubiquitous and distributed computing. Agents are software systems that model human social norms and conventions and autonomously solve problems. They are designed to run independently on behalf of humans or other agents.

> *"An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives."*
>
> *Wooldridge and Jennings 1999*

Measuring trust and reputation is intrinsic to the process of decision-making in intelligent agents. Although they can work in conjunction with other agents, their goal may have requirements that lead them to compete with other agents for limited resources. Since the environment is entirely open, an agent may encounter other agents that are uncooperative or even deceptive. The *benevolent agent assumption* [5] is not applicable in such an environment. With no guarantee of reliability, an agent in a complex multi-agent society needs to be able to determine which agents it can trust.

Whilst trust is an intuitive principle in human societies, in a computer system it must be quantified in order that it can be expressed. Sabater i Mir proposes a trust mechanism named ReGreT [8] that calculates not just trust values, but also reputation and credibility values, providing a degree of reliability for each. An agent may then use this data in decision-making. To achieve this sophistication, ReGreT takes into account direct experiences and information from other agents, and social network analysis. It considers several types of relation between agents, and uses these in the calculation of credibility and reputation. Furthermore, ontologies are used to reason about different types of related reputation. For example (taken from page 43 of thesis), it can be assumed that the reputation of being a good airline summarises the reputation of having good planes, the reputation of never losing luggage and the reputation of serving good food. In turn the reputation of having good planes summarises the reputation of having a good maintenance service, and the reputation of frequently renewing the fleet.

Sabater i Mir's analysis is comprehensive. He presents a testing framework named SuppWorld, which models a supply chain. This is then used to investigate the effectiveness of agents that use the ReGreT system, comparing with several other trust frameworks. The study concludes that the addition of witness information and social network analysis can radically improve the performance of an agent.

The Semantic Web Agents Project (MIND SWAP)[5] is also coordinating key research in the field of agent-based trust mechanisms.

[4] http://www.epinions.com

[5] http://www.mindswap.org

Trust within Distributed Networks and DDTM

Creating security systems within large distributed networks such as the Internet is problematic. Individuals are hard to uniquely identify, and a unified trust system must be extremely scalable and flexible. Enforcement of a uniform access control policy is not suitable for the large number of entities in a distributed system. Instead, Lei and Shoja propose a system that uses a tree of trusted peers to delegate trust decision making through the network [6]. For each pair of entities, three numerical measures are calculated by the proposed Dynamic Distributed Trust Model (DDTM): the direct trust value, indirect trust value and the authorisation level.

- The **direct trust value** is calculated by measuring performance (a measure of reliability, based on the proportion of transactions with a successful outcome), and loyalty (a value which increases after a threshold amount of successful transactions).

- If an entity a directly trusts another entity b, the **authorisation level** is a measure of how much entity a trusts the recommendations made by entity b.

- The **indirect trust value** is a measure of how much an entity trusts another unknown entity, calculated using a Dempster-Shafer reasoning algorithm that attempts to model uncertainty by a degree of belief. Subjective parameters are used, such as risk, indemnity (trust is recommended by a third party) and inclination to suspicious usage patterns.

The DDTM is designed to authorise access to content within a distributed environment. A content owner gives access to a node he directly trusts, and allows this node authority to delegate access to others. This trusted node becomes the **root** of a Trust Delegation Tree. Two types of certificate (signed message) are issued in order to expand the tree and distribute the responsibilities of the root:

- A **delegate certificate** is used to delegate authority from one entity to another, and contains the full path from the root to itself in order to verify its trustworthiness. The certificate is valid for a finite time period.

- A **recommendation certificate** is used to authorise a requestor, and contains the delegation chain. This is then authorised by the content owner, who checks that the delegate chain is valid.

A supporting protocol (Dynamic Distributed Trust Protocol) is specified, which supports the process of certificate submission and verification, and updating of the TDT (Trust Delegation Tree). The protocol has been validated by the SPIN model checker.

Lei and Shoja's proposal represents a detailed mechanism for evaluating and propagating trust through a distributed network. The system is interesting as it is generic, and could be applied to many different application domains on the Internet. However, it does not seem truly scalable. The advantages of distributing authorisation

responsibility seem to be largely negated by the reliance on the content owner for verifying the trust certificate chain.

It would be interesting to see an applied DDTM model to test performance, and also to test the accuracy of ratings using control data.

Semantic Web

> *"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."*
> Tim Berners-Lee, James Hendler, Ora Lassila, The Semantic Web, Scientific American, May 2001

The Semantic Web [1] is a term used to describe many different concepts. The overall aim is for pervasive meta-data in documents, allowing machines to locate and reason about content. Ideally software agents will do the hard work of locating resources on a distributed network (ie the Internet), allowing humans to concentrate on the content.

The W3C (World Wide Web Consortium) is responsible for standards such as OWL (the Web Ontology Language) [11] and RDF (Resource Description Framework) [12]. These standards are means of sharing semantic data. RDF is a form of XML (eXtensible Markup Language, a text markup language for the interchange of structured data). The semantics of RDF can be defined for a specific application domain. Triples of subject, verb and object are defined, and collections of these triples form ontologies. In the context of distributed networks, ontologies are shared vocabularies that give meaning to data. Uniform Resource Identifiers (URIs) are a fundamental component of the current Web and are a foundation of the Semantic Web.

The Semantic Web is an evolving and enabling technology. In the context of the recommender system it offers an extensible data format for storing reputation and social network data. It may also allow integration with existing social networks (see FOAF Project). Berners-Lee predicts that URIs will “break out of the virtual realm” and start to point to real-world entities such as mobile phones and TVs. As the Semantic Web begins to integrate everyday life with computer systems, the proposed system for making trustworthy recommendations (i.e. “digitising word-of-mouth”) will enable software agents to make far better choices on behalf of their human counterparts.

FOAF Project

As a means to store social network data, the FOAF (Friend of a Friend) project is a Semantic Web vocabulary that aims to build machine-readable social network data into the World Wide Web. It was originated by Dan Brickley and Libby Miller. The principle types of data stored are:

- Profile information (name, email address, employer, homepage etc.)
- Relationships with other individuals

Individuals are identified by email address. In order to avoid harvesting by spammers, email addresses are often concealed by a hash code generated by the sha1sum algorithm. An email address can be mapped to a sha1sum hash, but not vice-versa.

Data is stored in RDF format. This is an example of the layout of a FOAF file:

```
<rdf:RDF
      xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:foaf="http://xmlns.com/foaf/0.1/"
      xmlns:admin="http://webns.net/mvcb/">
<foaf:PersonalProfileDocument rdf:about="">
  <foaf:maker rdf:nodeID="me"/>
  <foaf:primaryTopic rdf:nodeID="me"/>
  <admin:generatorAgent rdf:resource="http://www.ldodds.com/foaf/foaf-a-matic"/>
  <admin:errorReportsTo rdf:resource="mailto:leigh@ldodds.com"/>
</foaf:PersonalProfileDocument>
<foaf:Person rdf:nodeID="me">
<foaf:name>Chris Beach</foaf:name>
<foaf:title>Mr</foaf:title>
<foaf:givenname>Chris</foaf:givenname>
<foaf:family_name>Beach</foaf:family_name>
<foaf:nick>Beachy</foaf:nick>
<foaf:mbox_sha1sum>ec28eb17fb7f3213ee0a610f7a2e13cced8f2cd3</foaf:mbox_sha1sum>
<foaf:homepage rdf:resource="http://www.chrisbeach.co.uk/"/>
<foaf:depiction
rdf:resource="http://www.chrisbeach.co.uk/images/uploaded/image5517.jpg"/>
<foaf:phone rdf:resource="tel:07971634090"/>
<foaf:workplaceHomepage rdf:resource="http://www.bath.ac.uk"/>
<foaf:knows>
      <foaf:Person>
      <foaf:name>Stephen Bellamy</foaf:name>
      <foaf:mbox_sha1sum>8bfe452af76581f7a13f41c3987d0a8f19ff13be</foaf:mbox_sha1sum
>
      </foaf:Person>
</foaf:knows>
<foaf:knows>
      <foaf:Person>
      <foaf:name>Alex Hliton</foaf:name>
      <foaf:mbox_sha1sum>7a90d1482a8603541dd8ed6b0e6ceeac59e02d7d</foaf:mbox_sha1sum
>
      </foaf:Person>
</foaf:knows>
</foaf:Person>
</rdf:RDF>
```

The proposed recommender system could interoperate with existing FOAF files, providing a test-bed of data. In this report we have seen TrustMail, an application of

this technology, which extends the FOAF vocabulary to include reputation data on individuals.

3 Feasibility and Expansion of Proposed System

The correlation between reputation and seller success in eBay has shown that electronic reputation ratings have an impact on consumers. The proposed system will further expand on eBay's successful mechanism by tying ratings to individuals within a social network. In doing so it will largely mitigate biased reviews from parties with a vested interest. Additionally, it will offer traceability for opinions, removing the veil of anonymity from current electronic opinion systems.

Choice of Architecture

The original project proposal suggested a peer-to-peer (also known as P2P) architecture be used to connect rating sources and establish the social network. Ratings would be propagated in a similar fashion to illegally-distributed MP3s.

In discussions with researchers at Bath the initial proposal has been changed and P2P architecture will not feature in the implementation. A P2P system would be reliant upon clients running on users PC's and would suffer overheads in making connections and handshaking between clients. In addition, IP addresses for other users must be found in order to make a connection. It was suggested that instant messenger systems might provide this information. However, further investigation of the popular MSN Messenger protocol showed that all communication between clients passes through central servers and client IP addresses are not exposed to each other. It is likely that the other IM clients work in a similar fashion.

So, the new proposal is that web-based server infrastructure will be used to store relationship and rating data. The system will remain decentralised and the data will be stored in the personal web-space of users. Since the initial community of users is likely to be within the academic and technical community, the requirement for personal web-space should not be an issue. To write and read the data, a server-side API can be implemented. This will allow data-access through the HTTP protocol.

This approach offers several advantages over P2P:

- Rating data will always be available, whether the user is currently using the system or not.
- Users can be uniquely identified by their personal website URL. This avoids the problem of 'ballot stuffing' votes from anonymous users.
- Communication between servers on the Internet is less likely to be restricted by firewalls than communication between clients.
- Autonomous software 'spiders' similar to those used by search engines may aggregate ratings data.

Choice of Web Browser

The following browsers are the dominant products in terms of market share (shown in brackets, figures taken from 2005 survey by onestat.com[6])

- **Internet Explorer** (86.63%) is Microsoft's web browser, packaged with all versions of their operating system since Windows 95 OSR-2.
- **Firefox** (8.69%) is an open-source browser developed by the Mozilla Foundation, which borrows from the Mozilla browser and former Netscape code-base. It may be downloaded free-of-charge.
- **Safari**, (1.26%) written by Apple, is packaged as the default browser of their operating system *OS X* (version 10.3 and later).
- **Netscape** (1.08%) takes several forms, the earliest being written by the now-defunct Netscape Communications Corporation. The most recent version has been written for AOL (America Online) by Mercurial Communications.
- **Opera** (1.03%) is a browser developed by Opera Software, which has an excellent reputation for security. It does not support extensions.

Internet Explorer is the most highly used web browser on the Internet at time of writing. However, Firefox is gaining market share due in part to its aggressive marketing campaign, which has included grass-roots publicity and a two-page advert in the New York Times. In the academic and technical community, usage of Firefox is much more common. W3schools.com (an online web development resource) reports that 24.6% of its visitors are Firefox users (as of May 2005). Opera and Netscape have negligible market share, and neither have shown growth in the last few months.

Firefox allows extensions to be written using text-based layout and scripting languages. It exposes a rich API to the extension developer, with access to the core browser components. There is no requirement for a proprietary development environment, unlike Internet Explorer, which requires code to be compiled into a proprietary .DLL (Dynamic Link Library) file or ActiveX control. Opera and Safari both have very limited support for extensions.

The browser plug-in will be developed for the Mozilla Firefox browser. Unlike Safari and Internet Explorer, Firefox is **cross-platform compatible** (though earlier versions of IE were released for UNIX and Apple platforms). Unlike Opera, it is available free-of-charge.

[6] http://www.onestat.com/html/aboutus_pressbox37.html

Choice of Server-side Scripting Language

- **ASP** (Active Server Pages) is a server-side scripting language introduced by Microsoft for use with its *IIS* (Internet Information Services) server. Its syntax is the same as that of VBScript but it has an additional library of built-in objects for use in generating dynamic web pages.
- **Servlets** are JIT (Just In Time)-compiled programs that handle HTTP requests and responses. They leverage the Java Servlet API. Servlets can be generated from JSP (JavaServer Page) files, which are templates containing program logic inline with HTML. Servlets are executed within a servlet container, such as Apache Jakarta Tomcat, which manages servlet deployment, session state and database connectivity.
- **PERL** (Practical Extraction and Report Language) is an interpreted language with excellent text processing features and a vast collection of third-party modules. PHP is prevalent in UNIX based systems and has also been ported to most major operating systems including Windows and Mac OS X. Critics argue that PERL's wide support for different dialects, paradigms and shortcuts leads to code that is difficult to read.
- **PHP** (PHP Hypertext Pre-processor) is a popular open-source interpreted language that is used for creating both dynamic web content and standalone applications using the PHP-GTK library. It is loosely typed, but supports object orientation. PHP runs on most major operating systems including UNIX, Linux, Windows, and Mac OS X.

The server-side component of the DSA system will be written in PHP. Like Firefox, the PHP server platform is cross-platform compatible and unlike ASP, it is available free-of-charge. The language of PHP is expressive, allowing object orientation, reflection and a vast collection of APIs. It is also very well documented online[7]. The documentation includes user-submitted examples and commentary.

PHP is enabled on the campus web servers, with web-space available to all students and academic staff at the University of Bath. Unlike Servlets, PHP is commonly supported by commercial web hosts.

[7] http://www.php.net

Building the Social Network

Semantic Web technologies may help automate and enrich the social network required to provide reputation ratings. Pujol suggests the following factors that indicate relationships between individuals [9]:

- Personal web pages
- Authorship of joint reports or documents
- Participation in projects
- Hierarchical structure in the community or organisation
- Sharing of physical resources
- Sharing of virtual resources such as news groups, forums etc.
- Email traffic

All of these factors can be easily represented and analysed using existing Semantic Web technologies. If the proposed system were aware of these relationships it would be able to construct a richer social network and therefore gain more reputation ratings. With more ratings, a statistically more accurate average can be calculated.

Reviewing Reviewers

Investigating the online opinion site Epinions has revealed another feature that can be naturally integrated into the proposed system. Since the subject of recommendations are web sites, and recommenders themselves are also identified by their personal web site, **reviewers can be rated** by the system.

When browsing to the homepage of a friend who uses the system, the software client would find the ratings data-file (since it has a standardised filename) and identify the friend as a possible source of ratings. With a single click on the browser plug-in client, the friend could be added to one's list of trusted peers. The simplicity of this mechanism would enable a social network to be built quickly by a user of the system.

Research in the field of collaborative reputation mechanisms demonstrates that reviewing reviewers is an effective strategy to increase the overall relevance of ratings [2, 13]. It is worth noting that the problem of "transitive trust" remains. If person A trusts person B and person B trusts person C, should it be inferred that person A trusts person C? There may be personal reasons why person B trusts person C (love, for example), and these may be incompatible with inferring the trust onwards to another person.

However, the idea of ratings on reviewers could enable an interesting technical feature in the proposed system. Frequent and accurate reviewers will become authoritative 'super-nodes,' who are characterised by having received a large quantity of high ratings. A spider could be deployed to identify such users and publish their URI on a central website. Thus the system client can immediately present a list of

‘super-nodes’ to new users of the system, avoiding the need for them to build a large social network of their own.

Not only would this solve the problem of initiating social networks for new users, the prospect of having one’s homepage address published on a ‘top-ten’ list would also serve as a tantalising reward for highly participative users.

4 Requirements

Since the development process is evolutionary, the requirements of the system have emerged throughout the project. However, the basic functional requirements for the system remain the same:

- The system should provide a reputation rating upon browsing to a website.
- The rating value should be represent the opinions of trusted peers
- The system should expose its reputation data via Web Services

These requirements will be further analysed in the specification section. Prior to this, more general issues will be considered:

4.1 Integrity/Trustworthiness of Data

The system will provide a mechanism to measure the integrity of commercial websites. If the system itself lacks integrity then it is useless. In the worse case, information provided by the system may be deceptive to the user and lead them to disreputable vendors. Therefore, it is important that the following conditions are met:

- The source of a rating cannot be anonymous. Each rating should be traceable in order that it can be (potentially) verified. Traceability should be straightforward to implement, since the network of users will already be socially connected.
- All ratings supplied by a user should be accessible to other users. With this data, when one user subscribes to the ratings of another user, it is an informed decision.
- A user must specify the source of ratings used by the client. The system will not have a pre-defined list of ratings providers, since this would distort the aim of using entirely social networks. The networks should be built based on real-world relationships.

4.2 Scalability

The system will operate on the Internet and it may develop a vast user-base. It must deliver an acceptable level of performance under conditions of heavy use. Therefore is must be designed in such a way that the computational power available may be increased to compensate for increases in load. If the system uses recursive calls over several clients or servers, maintaining performance will be an important concern as the social network grows.

4.3 Security

There are many individuals who would have an interest in breaking the socially distributed nature of the system. For example, stakeholders in an online service provider would stand to gain from saturating the system with positive ratings for their own site and negative ratings for competing service providers. To do so they would have to affect the ratings of other people, since in normal system operation their own ratings will only be visible to individuals that have subscribed to them.

The system must be impenetrable to outside interference in order to preserve its reputation and usefulness.

4.4 Interoperability

Reputation data in the system shall be made available to other systems. Discussions with members of Bath's @lis research team revealed a possible application of the system with an agent-based project to deliver personalised tourism/cultural information (further details in the *Conclusions* section). SOA (Service Oriented Architecture, also known as Web Services) is the preferred technology for interoperating with the agent framework.

4.5 Performance of Client

The DSA client should not have a detrimental affect on the user's browsing experience. It should load and display rating data quickly, and should not significantly affect the time taken to load a webpage.

4.6 Portability

Users of the system will be running various different operating systems, including Linux, UNIX, Windows and Mac OS X. They will also be operating on different platforms, including x86 machines and Macintosh computers. The choice of implementation language and APIs should take this into account.

Rather than coding different implementations for different platforms, a single code-base written for a multi-platform framework will save development time and improve maintainability.

4.7 Usability

The system should be easy to use and the user should not require detailed usage instructions. Where possible, context-sensitive help should be included.

4.8 Reusability

The system should be constructed in a modular fashion, and 'coupling' between component parts should be kept to a minimum

4.9 Reliability

Since the DSA client will interoperate with the user's web browser, a fault in the DSA client may affect web browser functions. Therefore it is especially important that the client is reliable and well tested.

4.10 Maintainability

The system should be written in accordance with published coding standards and well commented. This will aid any further development and debugging.

5 Specification

A successful implementation of the DSA system should demonstrate the following behaviour:

5.1 Phase 1 (Basic Ratings)

1. Upon browsing to an online service provider (e.g. a website selling holidays, or the website of a broadband provider), a reputation rating of the site will be shown within five seconds.
    1.1. The displayed rating will be one of six icons (see *User Interface Design* in the *Design* chapter), or blank if no ratings have been found
    1.2. The icon representing the lowest trust rating should be animated in order to draw attention to the bad reputation of the site
2. The client should allow a positive (*thumbs up*) or negative (*thumbs down*) rating to be given to the website currently being browsed.
    2.1. Only one click should be required for a positive vote. A further click will open a dialog, where the user may write a short review (up to 255 characters).
    2.2. When clicking to make a negative vote, the review dialog is opened after the first click.
3. The rating / review of a site may be modified by the user when returning to the website.
    3.1. A click on the same vote option will (re)open the review dialog, with an option to edit the review given previously.
    3.2. A click on the alternative vote option will change the saved rating for the site. The review dialog will be shown if the new vote is negative.
4. The review dialog will allow a rating/review to be deleted.
5. A website with a particularly bad rating (a negative:positive vote ratio of at least 5:1) should be indicated distinctly by the client. This will be referred to as a '*risky'* rating in documentation.
6. Upon browsing to a website which has existing reviews, this should be indicated to the client.
7. The user should be able to view at least five existing reviews for the website with a single click or mouse hover.
    7.1. These reviews should be sorted in order of trust (e.g. taking into account the number of "hops" along the social network)
    7.2. The user should also have the option to view more reviews

8. Sites may be repositories for DSA rating data (e.g. personal homepages of DSA users). Upon browsing to such a site, the client should be given the option to subscribe to the DSA rating data it provides if a subscription relationship does not exist already.

    8.1. The user should be able to browse a list of ratings available on the server before (or after) subscribing to that server.

9. The client should be able to subscribe to up to 10 DSA rating providers.

10. The DSA rating server should allow up to five levels of recursion when searching ratings from other servers.

11. Online and context-sensitive documentation should be made available.

12. The DSA server will provide the following Web Services in order to interoperate with other software, including agents within the @lis Technology Net:

```
Rating[] getRatings( String domain, String requestChain )
```

- This method will return an array of Rating objects (see *Data Structures*).
- The `requestChain` argument will hold a semicolon-delimited string list of servers. These are the DSA servers through which the request has passed during the aggregation process. In general, this argument is used internally within the DSA system and may be supplied as an empty string when called from another system.

```
void addRating( Rating rating, String password )
```

- This method takes a Rating object and password string.
- If the password is found to match the server's password, the rating will be saved to the server. If not, an error will be thrown by the method

```
void changePassword( String oldPassword, String newPassword )
```

- This method takes two strings. `oldPassword` should be set to the existing password of the server (which will be an empty string if the server has been freshly installed)
- If the `oldPassword` value is found to match the server's existing password, the server will set its password to the value of `newPassword`. If the old password is invalid, an error will be thrown by the method

5.2 Phase 2 (Classification-Specific Website Reviewing)

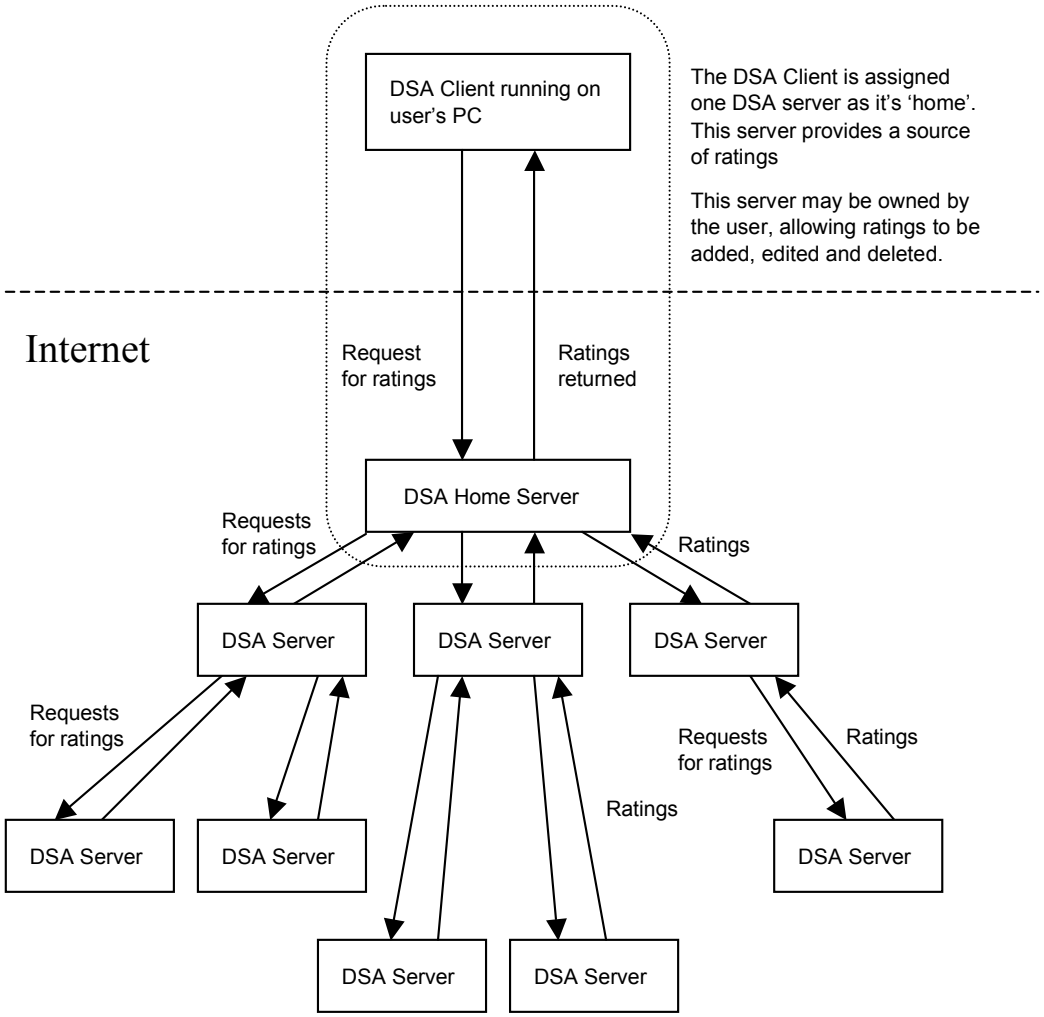
1. Upon browsing to a website, its classification should be determined where possible.
2. At least five classifications should be entered into the system as test data, and the DSA client should be able to correctly identify sites of these classifications with 90% accuracy.
3. Up to five additional criteria for review may be presented for a site of given classification.
4. The classifications and criteria will be stored in such a way that they may be updated by the client.
5. Classifications should be standardised. A mechanism must exist to merge classifications that are very similar.
6. When placing a review against these additional criteria, a scale of 1 (unacceptable) to 5 (excellent) should be used.

6 Design

Overview

The Don't Shop Alone (DSA) system will comprise two components: a web browser extension (the DSA client) and scripts that operate on a web server (providing the DSA server).

- The **DSA server** application will store ratings and reviews. It will also contain logic to aggregate ratings and reviews from other DSA servers. The system does not rely on a single server; it is not a conventional client-server application. Instead, users who have access to web-space may host a DSA server that will store their ratings. It will also store a list of other DSA server addresses (hosted by trusted individuals) from which to aggregate ratings.
- The **DSA client** application will run within the web browser and allow the user to view, add and edit ratings and reviews of websites. Initially, the client will not be configured to access a server. Upon browsing to a DSA server, the client will offer the option to subscribe to ratings and reviews provided by the server. If the server is owned by the by the user, the DSA client will allow ratings/reviews to be added, edited and removed (see Security Considerations for further details on the protection



Security Considerations

The system must be impenetrable to outside interference in order to preserve its reputation and usefulness, as discussed in the Requirements Specification chapter. The social network used by the DSA system should represent the real-world links that exist between the user and other individuals. With no central server, the effects of malicious hacking would be localised within a social network cluster and effects on the system as a whole would be mitigated.

Security was a major factor in the choice of protocol.

HTTP Protocol

The HTTP protocol is traditionally used for requesting and receiving webpages. It communicates using IP (Internet Protocol) packets over TCP (Transmission Control Protocol). Although HTTP supports user authentication, this requires a degree of authority on the web server that many DSA users will not have. On Apache web servers, a configuration file named `.htaccess` can simply be placed in a directory to protect it. On other platforms, however, the procedure is more complicated. Enabling authentication has the further drawback of requiring additional interaction by the user when installing the DSA server. It is therefore not viable for this system.

FTP Protocol

The FTP protocol would offer a convenient layer of protection when publishing ratings to a DSA server. It supports user authentication and is widely trusted by web developers. The user could supply the username and password for an FTP server, which would then be used by the DSA client to publish ratings onto the Internet. If the user did not have access to an FTP server they would still be able to use the DSA client in a limited capacity (subscribing to other people's ratings but not able to publish ratings).

FTP has obvious advantages, but unfortunately it will not be implemented. Development of the prototype DSA client exposed the lack of support for FTP in the APIs of Firefox. The only way to implement FTP support would be to write an interface that communicated at the socket level and parsed the FTP protocol. This is not viable given the time constraints on the project.

Reconsidering the HTTP Protocol

As a compromise solution, ratings will be published using the HTTP protocol, but authenticated using a password set once on the DSA server.

The DSA server stores its password in a file with a .php extension. Opening and closing PHP tags and comment tags surround the preference data. Therefore anyone not local to the PHP host (i.e. all Internet users) trying to access the data will get a blank response (the PHP server tries to pre-process the data but only finds comments so returns nothing). However, the DSA server is able to interogate the preferences since it can access the file without the PHP pre-processor kicking in.

Upon installing a new DSA server, the client toolbar prompts the user for a password to assign to the server. In its unprotected state, the server will save this password down to a PHP file as described above. The server will then block any future ratings submissions that are not accompanied by this password.

Sequence Diagrams

User Browses to Webpage

1. Get the domain of the webpage
2. Is the webpage on the same domain as the previously browsed page? If so, stop
3. Are there sufficient up-to-date ratings for the domain cached by the DSA client? If so, display the cached ratings and stop
4. Client sends the domain string to the user's DSA home server
5. DSA home server sends domain string to all known DSA servers
6. DSA servers respond to home server with ratings
7. Aggregated ratings are sent from DSA home server to DSA client
8. Client displays rating.
9. Stop

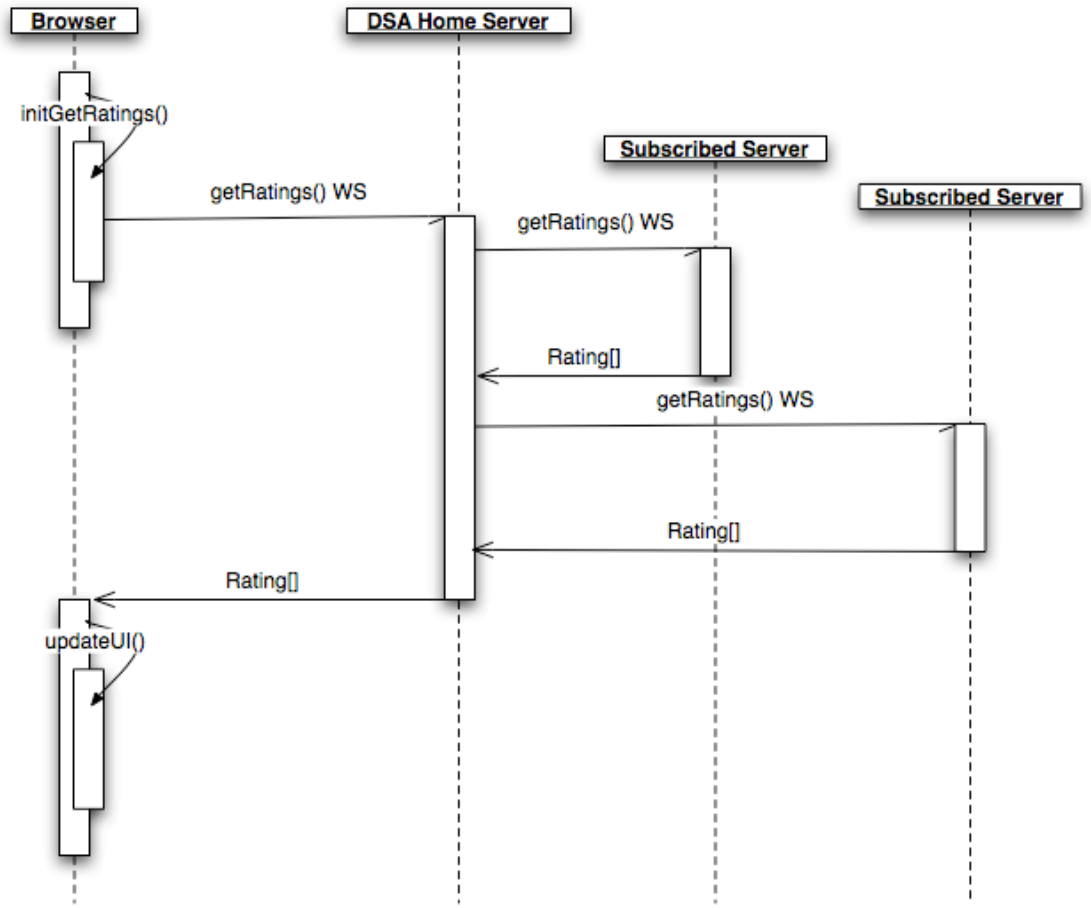


Figure 2 – Sequence Diagram for Steps 4-8

User Places Rating

1. a) If placing a negative vote, the DSA client prompts the user for a short review.
1. b) If placing a positive vote, the DSA client only prompts the user for a short review on the second click on the toolbar button.
2. DSA client calls `addRating()` method of the home server
3. DSA server checks password
4. a) Password incorrect, error returned to DSA client
4. b) Password correct, DSA server saves domain, rating and review to the resource file.
5. Stop

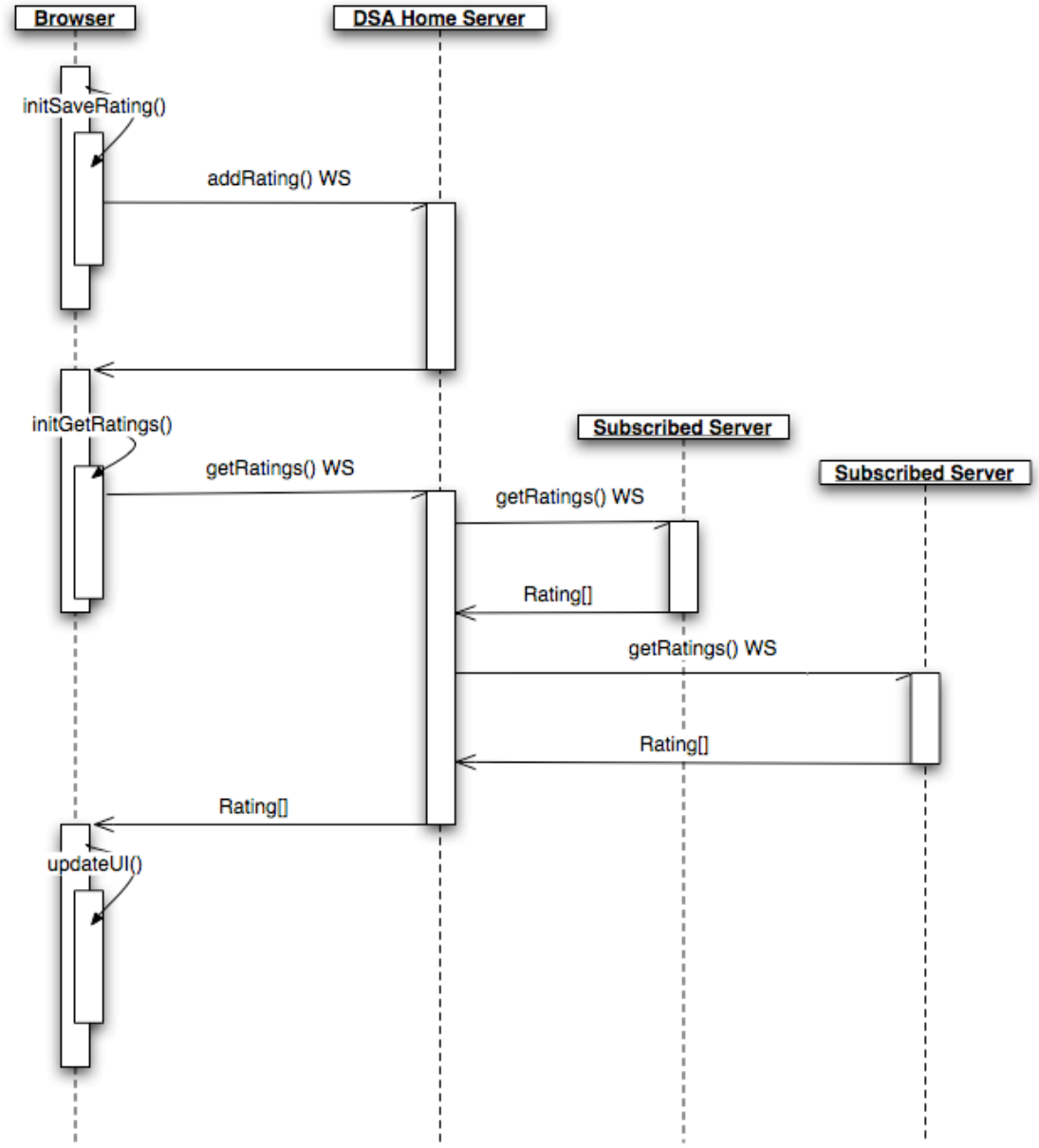


Figure 3 - Sequence diagram showing steps 1-5, with correct password and no review entered

Data Structures

This section specifies the attributes of objects/data structures that are passed through the system.

Request for Ratings

Name	Type	Notes
domain	String	The domain to be rated
requestChain	String	A semicolon-delimited list of servers from which the request has passed

The DSA client will send the domain as a string eg. “onlineshop.co.uk,” and the requestChain will be null initially. When a server forwards a request for ratings, it will add its URI to the requestChain list.

Response to Request for Ratings

The response will contain an array of Rating objects, with the following attributes:

Name	Type	Notes
domain	String	The domain to be rated
rating	Integer	-1 for negative, 1 for positive
review	String	Up to 255 characters of optional free text elaborating on the rating
serverChain	String[]	The list of servers from which the request has passed (the first in the list will be the source of the rating)

Client-Side Ratings Cache

Name	Type	Notes
domain	String	The domain to be rated
rating	Integer	-1 for negative, 1 for positive
review	String	Up to 255 characters of optional free text elaborating on the rating
sourceServerAddress	String	The address of the server from which the rating originated
downloadTimestamp	Date/Time	The time at which the rating was received

To minimise unnecessary traffic and load on the server, the client will store ratings and comments that it has received. The DSA client records a timestamp against ratings downloaded from the DSA server. This timestamp represents the time of download.

When browsing to a domain, the client-side cache will be checked before a request for ratings is sent to the server. If client-side cached ratings exist for the domain, they will be used and no request will be sent to the server. If any of the client-side ratings are found to be older than a threshold age, the request for ratings is sent and cached ratings for the domain are replaced with the new ratings received

Aggregation and Rating Display

The aggregation process described in the previous section will supply the client with an array of Rating objects (see previous section for the structure of a Rating object).

The client should summarise this data in a form that gives the user a rapid indication of the reputation of the site.

Several properties can be considered when weighting the importance of ratings. The design of this process should also consider the aggregation of ratings from social networks with various different topologies.

Social Network Topology

In most cases the connections between DSA servers do not form a simple tree-like "fan-out" from the user. There are likely to be cycles in graph, as shown in this example:

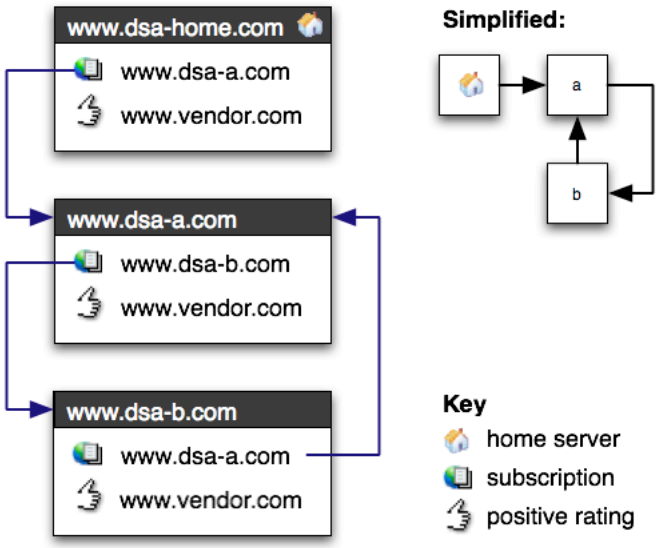


Figure 4 - A cyclic DSA server network

In this case, the requestChain variable of the request for ratings (see Data Structures) serves to prevent loops within recursion. The request will not be forwarded to a server that it has already passed through.

In this case, the connection from dsa-b to dsa-a will be effectively ignored by dsa-home's aggregation. The request from dsa-home would pass through dsa-a, dsa-b and then be prevented from passing to dsa-a again.

Without this prevention, infinite loops would occur. Servers connected in cycles would pass requests endlessly round the cycle.

Now we shall consider the following case, where dsa-home subscribes to both dsa-a and dsa-b:

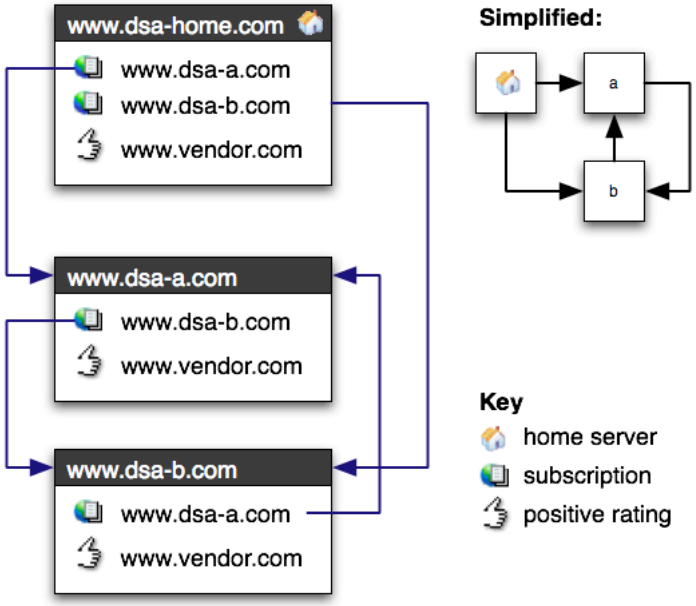


Figure 5 - A doubly-subscribed cyclic graph of DSA servers

This case raises a question: how many times should the ratings from dsa-a and dsa-b be considered?

Sending a request from dsa-home to dsa-a will return two ratings, as will the request from dsa-home to dsa-b. However, there are only two original ratings. Here we have two possible courses of action:

- Cull duplicate ratings keeping the version that came from the node closest to the requestor on the graph.
- Consider all four ratings. The dense linkage in this graph indicates that the individuals represented by the nodes form a well-connected group (perhaps a family, project team or a clique of friends).

In the interests of mirroring the real-world process of opinion sharing, the latter option has been selected. Opinions within a close-knit group should naturally carry more authority than those of a disparate individual at the outer region of the graph

Recursion Limit

If the social network of a user is large and well connected, rating requests will take longer to propagate. In order to prevent performance issues, an arbitrary limit shall be drawn on the aggregation process. A rating request will not be passed through more than five DSA servers.

Weighting the Ratings

Some ratings come directly from servers chosen specifically by the user. These servers are likely to be owned by individuals with a relationship to the user in the real world. Such individuals are likely to be highly trusted by the user. Other ratings come from servers connected to these subscribed servers, and servers connected to those, and so on. With each 'hop' from server to server, the user is less likely to know the owner of the server.

Therefore, the weighting of a rating is assigned based on the number of servers between the author and the recipient user. The weight of a rating is halved for each DSA server it has passed through before reaching the user.

A rating received from the home DSA server is assigned a weighting of one, so its weighted value is either 1.0 (a positive rating) or -1.0 (a negative rating). Ratings received from DSA servers subscribed by the home server are given a weighting of 0.5. Servers subscribed by those are given a weighting of 0.25 and so on.

Rating Value Icons

The following icons are used to display the rating of a site. The selection of image is determined by the ratio of total positive weighting to total negative weighting. Note the numbers are completely arbitrary. The thresholds may need to be modified depending on the usage of the system in production. For example, if people are inclined to make many positive votes for websites, the thresholds may need to be raised. If, on the other hand, people do not tend to give positive votes (in spite of good service), the thresholds may need to be lowered. Without this calibration, the rating icon shown will always indicate an extreme.

Ratio (positive:negative) Greater Than	Image
50:1	
20:1	
10:1	
5:1	
1:5	
Ratio (positive:negative) Less Than	**Image**
1:5 (risky)	

The lowest rating icon may only be displayed when 3 or more ratings have been placed.

The client should highlight an especially risky site. The icon that represents the lowest rating (shown above) should be animated such that it is more obvious. The animation should be a subtle increase/decrease in the brightness of the graphic.

User Interface Design, First Iteration

The extension will display a toolbar in the browser as follows:

The toolbar has been designed to be unobtrusive since it consumes part of the website "real estate". Rather than filling the toolbar with buttons, only three clickable areas are made available. This reduces the visual impact and complexity of the client, although further options are available by clicking the reviews button.

Colours are largely shades of grey rather than solid black or vibrant hues. The icons are simple and shadowed to match the style of other toolbars.

Review Menu

The review menu shows a selection of reviews, and two other items: 'Options' and 'Help'

In order that ratings are distinct, and separated, they are given background colours that alternate between two shades of blue. This helps differentiate them from menu

commands. There is a line separator between the ratings and the other two menu options.

Help

The help option on the context menu opens online documentation in the web browser. Such web-based documentation will be easier to keep up-to-date, and allow a mechanism for users to give direct feedback.

The help webpages will be hosted at http://www.dontshopalone.co.uk, and use the content management system written previously by the author of this project. The system allows content to be edited using a web browser, and also allows users to write comments that are displayed inline the content.

User Interface Design, Second Iteration

Whilst creating a reusable prototype and consulting with expert users, several improvements were made to the original user interface design.

- The **Help** elements did not belong in the Reviews menu. A **DSA** menu with a **Help** option was created on the left hand side of the toolbar. In a right hand side position, the menu would be visibly shifted when the reviews and ratings elements of the toolbar became shown or hidden.
- Wording was changed on negative voting button. An expert user commented that the original wording was negative and not intuitive enough.
- The relevant rating button is emboldened to represent the user's vote for the current website
- A logo has been added in order to give identity to the toolbar and make it more distinctive.
- The background colour of items on the reviews menu indicates whether each review is positive (pale green background) or negative (pale red background)

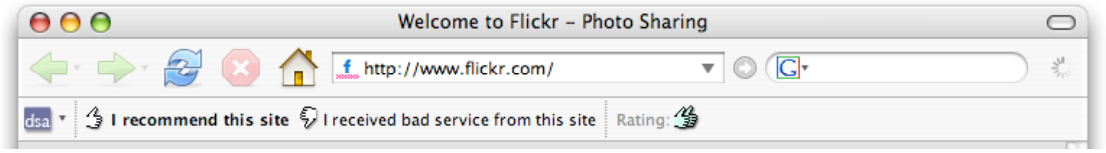


Figure 6 - Toolbar showing a positive rating

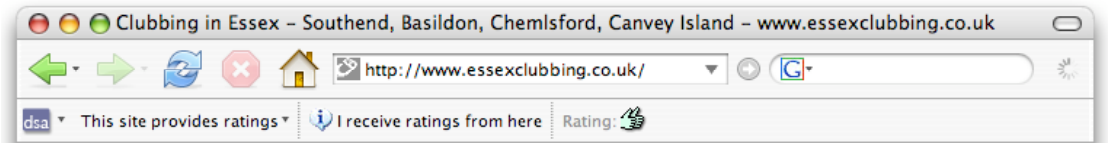


Figure 7 - Toolbar showing a subscribed DSA server

Figure 8 - Toolbar showing a home server

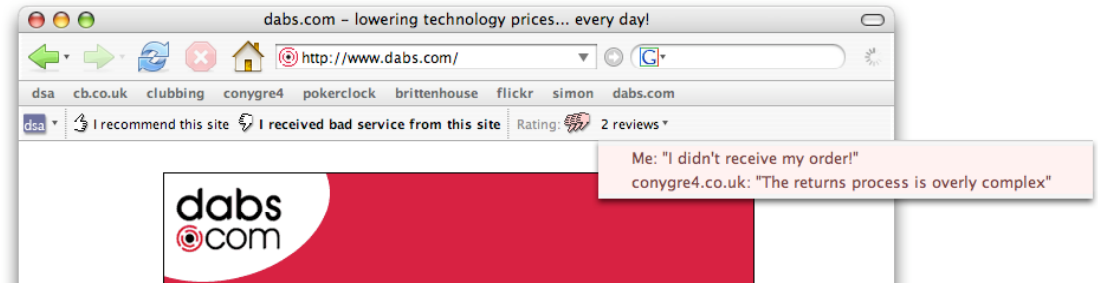


Figure 9 - Toolbar showing review menu

7 Implementation

The Don't Shop Alone (DSA) system comprises two components: a web browser extension (the DSA client) and scripts that operate on web servers (the DSA server). This section of documentation will detail the implementation of the extension and scripts.

Firefox Extension Framework

Extensions for Firefox (also referred to as addons or plugins) are written to a documented set of standards. This ensures they operate with the Firefox's built-in installation and update service. An online tutorial[8] provided much of the following information.

Extension source code is zipped into in an .XPI file. The code is divided as follows:

- Program logic is coded in JavaScript
- User interface is described in XUL (see *Client User Interface*)
- Two installation configuration files are written, one in RDF and one (for legacy versions of Firefox) in JavaScript.

Unique Identification for an Extension

Hundreds of extensions have been written for Firefox. Each different extension is keyed with a globally-unique identifier (GUID) by its developer. This helps Firefox to ensure that only one version of an extension is installed at any time.

An online tool[9] has been used to generate the GUID for the DSA client.

Overlays

Overlays[10] are XUL user interface files that allow the user interface of the browser to be extended in several ways. For example, a toolbar may be added or new options may be introduced into a menu or context menu. XUL also allows the reuse of standard browser components such as the auto-completing address bar and the various navigation buttons.

Extension Installation

The XPI installer can be loaded directly by the Firefox browser to begin installation. The extension is then registered by the browser and the source code is copied into the "chrome" directory within the user's profile.

Upon a restart of the browser, the overlay of the extension user interface is activated.

---

[8] http://roachfiend.com/archives/2004/12/08/how-to-create-firefox-extensions/

[9] http://extensions.roachfiend.com/cgi-bin/guid.pl

[10] http://www.mozilla.org/xpfe/xptoolkit/overlays.html

Extension Updates

In order to check whether an extension is up-to-date, Firefox polls its homepage on the Internet. If a newer version of an extension has been released, the browser will indicate this to the user and automate the update process. To facilitate this feature, the .RDF installation configuration file of each extension contains a URI that points to a Internet server hosting the latest version of the extension. For the DSA client, the URI is:

http://www.dontshopalone.co.uk/update.rdf

The file `update.rdf` contains two important details:

- The current version number of the extension
- The location of the .XPI file which contains the latest extension code

```
<?xml version="1.0"?>
<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
       xmlns="http://www.mozilla.org/2004/em-rdf#">
<r:Description about="urn:mozilla:extension:{37d4fa3f-0f09-4631-92eb-e0f60e474238}">
  <updates>
    <r:Seq>
      <r:li>
        <r:Description>
          <version>0.0.1</version>
          <targetApplication>
            <r:Description>
              <id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</id>
              <minVersion>0.8</minVersion>
              <maxVersion>1.2</maxVersion>
              <updateLink>http://www.dontshopalone.co.uk/dsa.xpi</updateLink>
            </r:Description>
          </targetApplication>
        </r:Description>
      </r:li>
    </r:Seq>
  </updates>
  <version>0.0.1</version>
  <updateLink>http://www.dontshopalone.co.uk/dsa.xpi</updateLink>
</r:Description>
</r:RDF>
```

Figure 10 - example update.rdf

Model-View-Controller (MVC) Paradigm

The first working prototype of the DSA client had several bugs related to the interaction between control logic and the user interface. These bugs manifested themselves as buttons unexpectedly disabling and enabling, options appearing and disappearing and general glitches in toolbar behaviour.

The problem was down to the disparate nature of callback functions used in asynchronous calls. Upon detection of a server, a callback function would suddenly execute in JavaScript, independent of any other running function. The same would happen for a rating response, or a rating save confirmation, or several other actions. All of these callback functions would make visual changes to the user interface. For example, if a DSA server was detected, the 'subscribe' button might be displayed. However, another callback could then conflict with this, requiring the 'subscribe' button to be hidden (if it was discovered that the server was already subscribed, indicated by a rating response). In the worst case, the conflicting updates to the user interface would leave it in an incorrect state. This is clearly unacceptable.

In order to resolve this issue, concepts from the Model-View-Controller (MVC) paradigm were employed. This aims to separate the user interface, the state model and the controlling program logic.

In the DSA client, the state of the currently browsed website is modelled in a single object, named `gSiteState`, which is instantiated in global scope when the user browses to a new domain. This object completely encapsulates the known state of the website, and contains functions that operate on this information to:

- indicate whether or not the site is subscribed
- indicate whether or not the site is the user's home server
- parse the rating array and `serverChain` string
- indicate whether the site state is displayable (that is, whether sufficient callbacks have been received)

There is a controller function called `updateUI()`, which uses the `gSiteState` object to update the entire user interface in one operation. This controller function is called at browser start-up, webpage navigation, and in each callback function. It only updates the user interface if `gSiteState` indicates an update is necessary.

Since control logic, user interface and state model were separated, the system works much more predictably and the code is decidedly neater.

Coding Standards

Maintaining code is easier if all developers adhere to a set of formatting and naming standards. Mozilla has published a list of standards for JavaScript[11], including:

- Prefixing global objects with a 'g' e.g. `gSiteState`
- Using upper case when naming constants e.g. `DSA_FILE`
- Beginning the name of a constructor with an upper case character

These coding standards have been used throughout the project.

Client User Interface

The extension user interface is written in XUL[12], a proprietary XML-based markup language created by the Mozilla Foundation. CSS (Cascading Style Sheets) are used to control the look-and-feel where possible. This approach centralises the formatting into a single file and allows a common theme to be used for components of the interface.

XUL documents have been created for the browser toolbar (in the form of an "overlay") and for the dialogs that request review and server password.

The principle tags used in the toolbar overlay are:

- `<toolbox>` is a container for a toolbar
- `<toolbar>` encapsulates a toolbar to be displayed under the menu (and other toolbars within the browser
- `<toolbarbutton>` represents a button with optional graphic to be displayed within a toolbar
- `<vbox>` is styled with CSS (`class="dsaSeparator"`) and used to create a vertical separator between elements in the toolbar
- `<menupopup>` represents a popup menu
- `<menuitem>` represents an item within a menu, which is usually assigned a JavaScript function in its `oncommand` attribute
- `<menuseparator>` is used to display a vertical line between menu items

Please see *Appendix A* for the XUL source code used to create the DSA client interface.

[11] http://kb.mozillazine.org/JavaScript_coding_guidelines
[12] http://www.mozilla.org/projects/xul/

Bridging the Client and Server

The interaction between the client (toolbar) and servers is fundamental to the implementation. The choice of protocol and technique will directly influence the performance and reliability of the system.

The two mechanisms used in the implementation are mature, proven technologies, described in more detail below.

Web Services

In order that the JavaScript client can communicate with the PHP-based server, a common protocol is required.

The HTTP protocol is used to request and receive web pages across the Internet. It also allows data to be passed in a request.

Web Services (sometimes referred to as **SOA** – Service Oriented Architecture) is a standard that uses the HTTP protocol to send and receive data in a well-defined form. An application server hosts a web service at a URI (universal resource identifier). An accompanying definition of its data structures and methods is published there in a form of XML called WSDL (Web Service Description Language).

Since WSDL is machine-readable, web services can be automatically discovered and interrogated. This makes the technology particularly interesting in the field of Multi-Agent Systems (MAS).

The major selling point of Web Services is that unlike other communication protocols (eg DCOM), they operate on the same port as web traffic are therefore generally not blocked by firewalls. Another feature of most Web Service implementations is support for **asynchronous** behaviour, which avoids the client from being halted whilst waiting for a response from the server. Such protection is required, particularly on the Internet, which offers few guarantees of bandwidth or connection reliability.

XMLHttpRequest

XMLHttpRequest is a proprietary technology first introduced by Microsoft into the JavaScript implementation of Internet Explorer version 5. Like Web Services, it allows scripts on a client to communicate asynchronously with scripts on a server.

XMLHttpRequest became a de-facto standard, and was implemented in the Safari browser on the Apple platform, and also in Firefox.

In the context of the DSA client, the XMLHttpRequest class is used to discover a server script and interrogate it to find its version, and whether it is protected with a password.

There are three principle methods and properties of the XMLHttpRequest object which have been used in the DSA client implementation:

- `open( method, URI, async )` method sets the server URI to receive the request.
    - The `method` argument may either be "GET" (for sending requests with simple parameters) or "POST"
    - The `URI` argument is the Web address of the server to receive the request. This may include parameters in a 'query string.' For example: `http://www.dontshopalone.co.uk/dsa.php?sniff=true`
    - The `async` argument is a boolean to indicate whether the call will be made asynchrously. If this is set to false, the program flow will be halted whilst waiting for a response.
- `onreadystatechange` property is set to a JavaScript function (referred to as the **call-back** function) to receive status updates as the request is fulfilled
- `send( content )` method initiates the request
    - The `content` argument is optional and may be used to pass an XML document or URL-encoded form data (when the method is set to "POST") within the request.

The call-back function assigned to the `onreadystatechange` property must have access to the XMLHttpRequest object. Therefore the XMLHttpRequest object is made global. As a response is received, the callback function queries several properties of the XMLHttpRequest object:

- The `readyState` property contains the phase of communication. Its initial value is 1 when the request is originally sent. When a well formed response has been received its value is 4. Note that the callback function is called for each of the four phases.
- The `responseText` property holds the text of the response from the server
- The `responseXML` property is similar to the `responseText` property except that it contains a XML DOM object when the response is in the form of XML. If the response is not in XML format, this property is unavailable.

PHP Server-Side

The DSA server has several responsibilities:

- Storing ratings and reviews for the user
- Fulfilling requests for ratings from either a DSA client or DSA server
- Sending requests for ratings to DSA servers

NuSOAP

In order to use Web Services, a third-party helper API called nuSOAP[13] has been leveraged. NuSOAP provides objects and methods to allow the creation and consumption of SOAP-compatible Web Services.

NuSOAP is mature project, and released under the Lesser General Public License (LGPL)[14]. This allows its use in any other projects, whether distributed as free software or otherwise. There are some restrictions in place but these apply more to the LGPL software than the software that leverages it. Under the LGPL, the only real condition for the DSA system is that it must be possible for it to be linked with a newer version of nuSOAP. Such non-static linking can be achieved simply using a PHP `include()` statement to incorporate the nuSOAP library file at runtime.

The Rating Object

The PHP script has been implemented in a partially object-oriented fashion. This helps to 'wrap' related functionality within the source code. It makes software reuse more practical and code more readable.

A class of object encapsulates a rating/review. It has methods for getting:

- an HTML representation (as a row of a <table>)
- a name-indexed array, suitable for transmission via the NuSOAP API
- a delimited string representation, suitable to save to a file

A class of object encapsulates a DSA server. It has methods to produce an HTML representation for the subscribed server list page.

[13] http://sourceforge.net/projects/nusoap/

[14] http://en.wikipedia.org/wiki/LGPL

Features Not Yet Implemented

FTP Support

In the original design, FTP was to be used as a secure means to publish ratings to the server, and also to install the server-side PHP script.

Unfortunately it was found that at time of writing, the support for FTP interaction in Firefox is very limited. There appears to be no documented API for the FTP protocol.

It would be possible to establish a connection and communicate at the socket level. However, implementing the FTP protocol is secondary to the aims of this project, and hence support has been dropped.

As described earlier in this chapter, the client is able to save ratings with a good degree of security over HTTP.

Phase 2

The classification-specific reviewing feature was not implemented due to time constraints.

Caching

Neither the client nor the server caches ratings at present. This feature will be important in a production release of the system. See *Direction for Improvement* section for more details

Minor Features

The following features have not yet been implemented. The specification number is indicated in brackets

- Screen for displaying more reviews (6.2), although the reviews menu currently shows all reviews
- Sorting displayed reviews in order of trust (6.1)
- Adding review for positive ratings (2.1)
- Removing ratings (a user may only change an existing rating from negative to positive or vice versa). This should be included as a button in the review dialog (4)
- Animation of the lowest rating icon (1.2)
- Imposed upper limit on the number of subscriptions that a DSA server will accept (9)

Features Additional to Design

Some technical features were added during implementation:

Build Script

A UNIX build script was written. This compiles the extension by compressing the source code and interface into a JAR file and then copying this file into the Firefox profile directory. It was found that this was sufficient to apply any The script then runs Firefox from the console, allowing debugging messages to be shown.

The build script aided rapid code development and debugging.

Automated Bug Reporting

Although debugging during development was difficult, the situation will be much improved upon release.

An error handler in the PHP server-side script saves a log (`phperror.log`) of all errors it encounters. Just like the ratings, this error log is accessible over the Internet. Hence the developer can get quick access to a verbose description of any problem when the system is in production (no user action required).

Debugging Mechanism

Debugging a program written solely in JavaScript or PHP is straightforward. However, when a JavaScript function is calling a PHP script via Web Services, the extra communication layer makes it hard to follow control flow from one function call to another.

This is made especially complicated when the JavaScript is calling PHP which then recursively calls more PHP scripts.

There are debugging mechanisms designed for this scenario. Remote debugging is a language feature that allows a desktop client to establish a TCP session with the application server in order to receive errors and warnings as they happen. However, this feature is unsupported in PHP 4.

Instead, a custom error handler function was built into the PHP script. This sent emails with error logging information, and also wrote this output to a file (`phperror.log`). This is an example of the output during development:

```
1024: getRatingsFromDSAServers: Checking server [http://www.chrisbeach.co.uk/] in
/homepages/15/d83177889/htdocs/dontshopalone/dsa.php [226]
(http://dontshopalone.co.uk/)
1024: getRatingsFromDSAServers: Checking against already-requested server
[http://www.chrisbeach.co.uk/] in
/homepages/15/d83177889/htdocs/dontshopalone/dsa.php [232]
(http://dontshopalone.co.uk/)
1024: Received request chain:
[;http://www.chrisbeach.co.uk/;http://dontshopalone.co.uk/;http://chrisbeach.co.uk/]
in /homepages/15/d83177889/htdocs/dontshopalone/dsa.php [191]
(http://dontshopalone.co.uk/)
```

Figure 11

Direction for Improvement

Caching

Testing shows that aggregation over several levels of recursion performs within timing specifications, and that caching is not required for the system under test conditions. However, in future, it will be crucial to implement this feature in order to reduce the load on well-subscribed DSA servers. In production, the DSA server may be hosted on slower, less reliable hardware, which will radically affect performance of the system.

Caching will also increase performance both client-side and server-side, and will allow the recursion limit to be increased. This will increase the number of rating providers that can be aggregated. A DSA server could be designed to store all the ratings of its trusted peers, therefore reducing the process of aggregation by one recursion level.

A recommended technique for implementing caching is a functional programming style called *memoisation*[15]. This preserves existing code but 'wraps' certain functions with a higher-order function that caches their results. For example, the client-side `initGetRating(domain)` function may be wrapped in a *memoise* function that will check a local table of cached ratings before making a call to the server. If a local result (from a previous server call) is found, then use of the results of this previous call will avoid a new call to the server.

Clearly the client should not permanently cache all ratings received from the home DSA server. This would result in stale data, which may have changed on the server-side. Cached data could be given a lifetime, after which it will be deleted. In the simplest case the cache will be flushed when the browser is closed.

Hash-Encrypting Passwords

It is vitally important that the saving of ratings to a DSA server is restricted to the owner of that server.

Therefore the DSA server is protected by a password that is initialised by the user after installation. The user's DSA client will then send this password to the server with every request to save a rating. Whilst this offers some protection, the password is currently sent in plain text form (inside the XML SOAP request over HTTP). Whilst very unlikely, this request may be intercepted by a third party. The interceptor would then be able to save ratings to the DSA server using the same password.

In order to protect against this, the password should be converted to a hash code before being sent. This process, sometimes called 'hashing,' is designed to mask the original password whilst retaining its unique 'signature'. A hashing function takes a string and generates another string, usually of fixed length, which is as unique as possible. The hashed password can then be verified by the server, which compares the hash codes of the sent password and the known password.

---

[15] http://en.wikipedia.org/wiki/Memoisation

This is clearly not yet secure, since the interceptor could merely send the intercepted hash code in future requests.

So, the combination of password, rating value and domain is hashed, rather than just the password. The server is able to verify the combination-generated hash code because it knows each of the original constituent parts. However, an interceptor cannot 'reverse engineer' the hash code back to the password, so will be unable to generate valid combination hashes for future requests. All (s)he is able to do is save the same ratings that user has already saved (duplicate values which will be ignored by the system).

Improving the Rating Weighting Algorithm

At present ratings are simply weighted by the path length and by the amount of clustering in the social network. There are many other important factors that should contribute to the authority of a rating, or a user of the system.

The age of the rating is inversely proportional to its importance. Opinions represent a instant in time, a mere snapshot of a situation. A rogue vendor is likely to remain a rogue vendor and previous ratings will therefore retain their deterrent significance. However, if a good vendor turns bad, old ratings would be deceptive, and it would take much new reputation data before the average rating better represented the new situation. Therefore, the perceived weight of a rating should be reduced with time.

The authority of a user could be measured by several criteria including:

- Links to his personal webpage, measured using the Google pagerank algorithm
- Authorship of joint projects, reports or documents. This data can be represented in FOAF format
- Hierarchical structure in the community or organisation. This data can also be represented in FOAF format
- Sharing of virtual resources such as news groups, forums etc.
- Email traffic, which would require interaction with a mail server.

As described in the Literature Survey, using such data would give context and weight to the links between individuals. It would also identify individuals with many incoming connections. Such users are most likely to be authoritative.

Visualisation

The graph structure of the social network may be navigated crudely by browsing between DSA servers, following subscribed server links. Much more could be done with the social graph data present in the system.

Visualisation of this structure would reveal a macroscopic picture of connections between DSA servers. In particular, two properties could be highlighted:

- The authority of nodes, measured by the number of incoming links
- The strength of a link between nodes, using additional data such as
    - Sharing physical resources, such as host
    - Geographical proximity
    - Email traffic

The graph visualisation could be built by a Macromedia Flash program, for example, which spiders the DSA servers, following subscription links and builds a graphic such as the following:

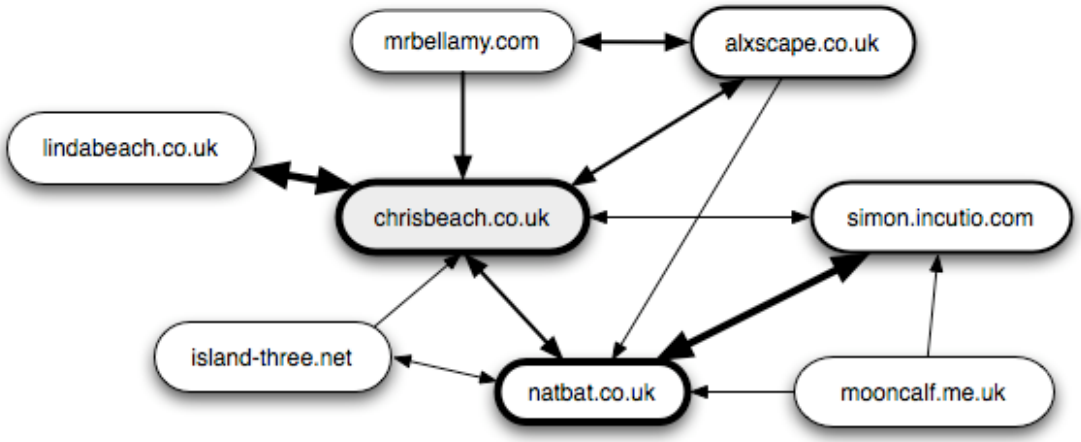


Figure 12 - Example visualisation

In Figure 12, the thickness of the connecting lines indicates the strength of the connection between nodes, taking into account external factors listed above. The thickness of the border around a node indicates its authority, based on the number of incoming links. Clicking a node would centre it on the screen and load its connected nodes

Comparing Similar Sites

It would be very useful for the DSA system to compare sites of the same classification. Automatically finding the classification of a site is a difficult task for a machine. Ideally there should be some user input into the process.

The DSA client could provide a mechanism for comparing the current site's rating with another similar site. This would provide a useful tool to the user, and could also capture pairs of similar sites for the purposes of the system. Therefore, given enough pairs, the system can construct a graph of related sites. Heavily connected nodes in

the graph would be sites of the same type (eg ryanair.com, easyjet.com and bmibaby.com). This would allow the user to get a 'top-10' list of related sites (eg 'Top 10 airlines'), sorted by rating.

FOAF Production

As described in the Literature Survey chapter, FOAF (Friend of a Friend) is a data format for expressing profiles and relationships of individuals. The subscriptions between servers indicate trust relationships between people, and to a large extent, the domain of a user's server is a unique identifier for that individual.

Therefore, it would be useful for the DSA server to output an RDF file with FOAF data in order that other systems can use this standardised data in mining operations. The Semantic Web will become much more significant when well-formed data like this becomes ubiquitous on the Web.

Protocol for Non-Human Users to Include Metadata

Currently, the review attribute of a rating allows the user to elaborate on the reasons behind the rating. Could this also be used for non-human users of the system?

Many agents implementations use RDF data to describe knowledge. Such data could be encoded into the review, for example:

```
<rdf:RatingRationale about='www.vendor.com'>
      <rat:service>Shipping</rat:service>
      <rat:issue>Wrong Address</rat:issue>
</rdf:RatingRationale>
```

Figure 13 - Example RDF rating metadata

This format allows software agents to parse the rationale behind a rating and make logical inferences or compare one rationale with another, like-for-like.

Integration with Del.icio.us

Del.icio.us[16] is a website that offers collaborative bookmarking of websites. It has been suggested that the DontShopAlone should publish positive-rated sites to a user's del.icio.us profile. This would save the user effort in publishing bookmarks.

It is also possible that del.icio.us could feed useful information back into the DontShopAlone system. Del.icio.us has a system for collaboratively tagging websites with arbitrary strings to indicate their nature. Such user-supplied metadata would be ideal for feeding the site classification system proposed in phase two of the design. A protocol would need to be implemented to merge, or standardise tags so that a definite, concise list could be constructed over time.

[16] http://del.icio.us/

8 Testing

The DSA client is designed to conceal the technical processes that lead to the display of a rating for the current website. However, for testing purposes, the browser has been loaded from a terminal session, which shows detailed debugging messages. Messages include:

- Sequence of function calls
- Time taken to receive and aggregate ratings
- Weighting of ratings received
- URIs used in Web Service calls / XMLHTTPRequest calls
- Values of state variables

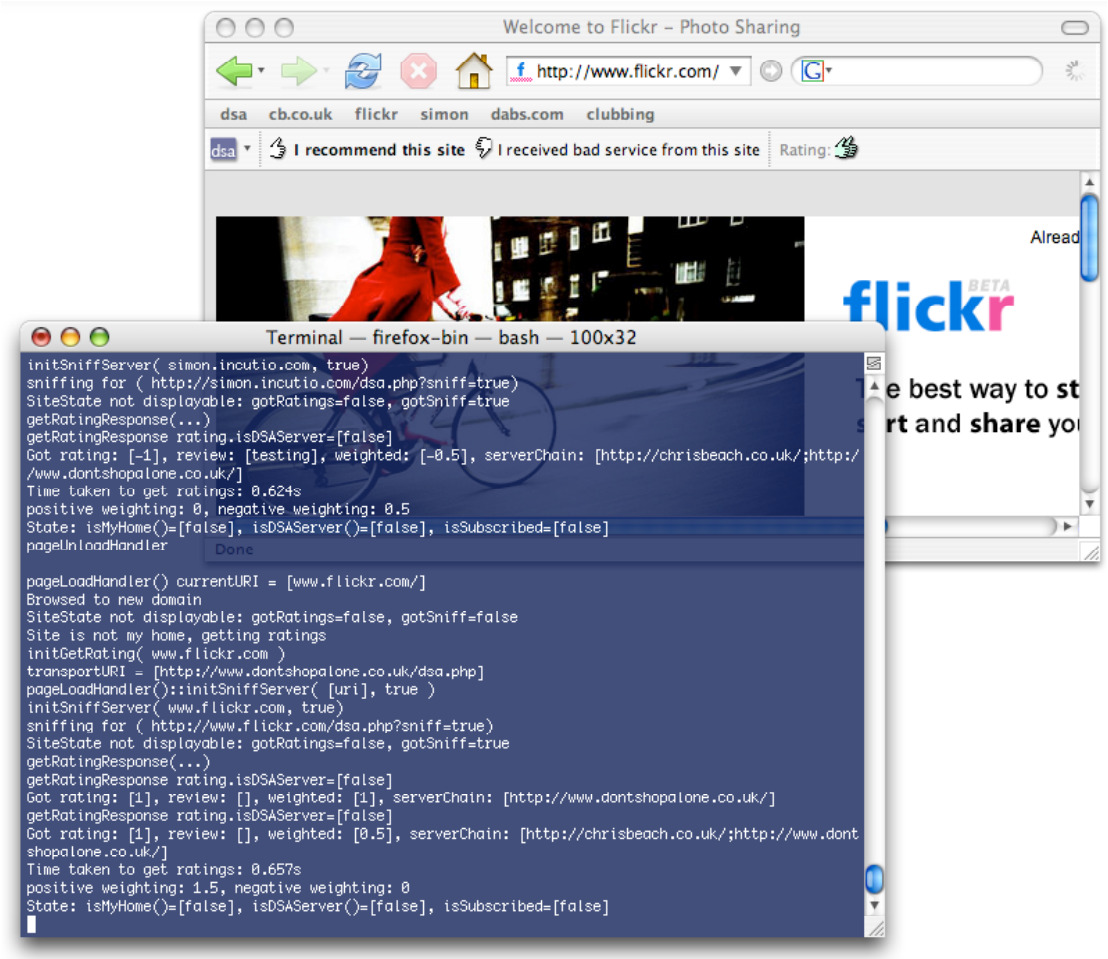


Figure 14 - The DSA client in debugging mode

Rating Aggregation Behaviour

Several cases have been chosen to test the aggregation algorithm. For a description of the design of the algorithm, please refer to the design chapter.

Care was taken to ensure uniform conditions (eg network infrastructure, browser, platform, network congestion, DSA server URIs etc) when timing the rating aggregation process.

No Home Server

When no home server is set, the client should display the following:

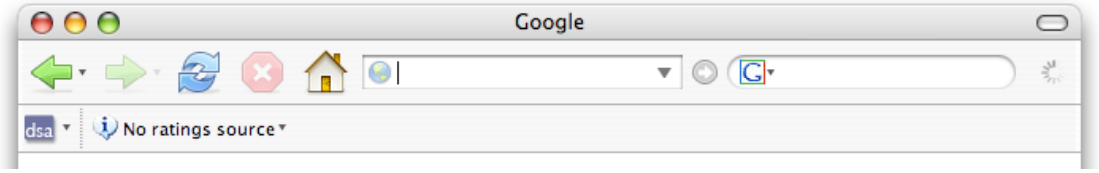


Figure 15 - No rating source

No ratings can be aggregated or saved since there is no home server set. This should be the case after a fresh installation of the DSA client (no previous installation). When the user browses to a DSA sever they should see the following:

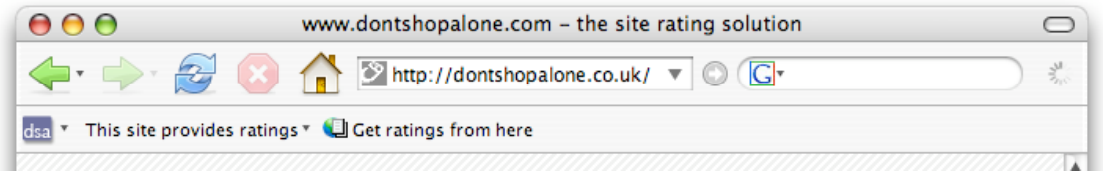


Figure 16 – Browsing to a rating source

The test results are as follows:

Test	Expected Result	Result
Fresh client installation, Mac OS X	'No ratings source' message shown, no other toolbar buttons	✓
Fresh client installation, Microsoft Windows XP	'No ratings source' message shown, no other toolbar buttons	✓
Browsed to rating source	'This site provides ratings' message and 'Get ratings from here' option	✓

Home Server Set

When a home server is set, ratings may be saved and loaded. This simple test case assumes the user has browsed to www.vendor.com and has previously saved a positive rating for it.

Key

home server

positive rating

In this case the client should receive one rating for www.vendor.com, and weight it with a magnitude of 1.0

Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.0	

A number of tests were carried out in order to measure the average performance:

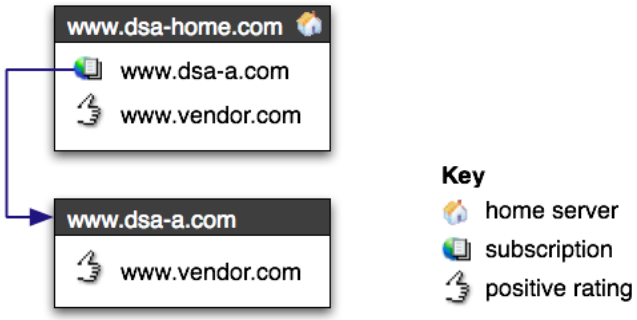
Test #	Time Taken (s)	Notes
1	0.40	First browse after browser startup
2	0.36	
3	0.34	
4	0.37	
5	0.35	
Average Time (s):	**0.36**	

Note that URIs described in the tests (eg www.vendor.com, www.dsa-a.com etc) are for illustration only. The actual tests were carried out hosting the DSA server on various domains including:

- www.chrisbeach.co.uk
- www.dontshopalone.co.uk
- www.essexclubbing.co.uk
- www.conygre4.co.uk
- www.pokerclock.co.uk

One Subscription

In this case a home server is set with one subscription. This test should demonstrate that a rating from a subscribed server carries half the weight of a rating from the home server



In this case the client should receive two ratings for www.vendor.com, weighted accordingly:

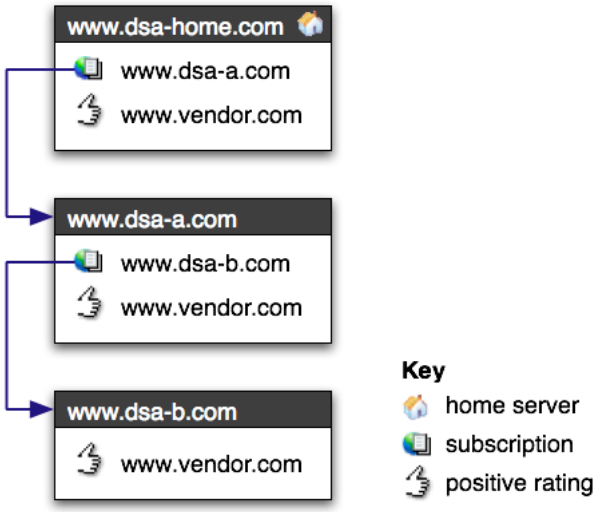
Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.0	✓
www.dsa-a.com www.dsa-home.com	0.5	✓

A number of tests were carried out in order to measure the average performance:

Test #	Time Taken (s)	Notes
1	1.08	First browse after browser startup
2	0.61	
3	0.64	
4	0.80	
5	0.51	
Average Time (s):	**0.73**	

Simple Recursive Case

In this case a home server is set with one subscription, and the subscribed server has a subscription to one other server.



In this case the client should receive three ratings for www.vendor.com, weighted accordingly:

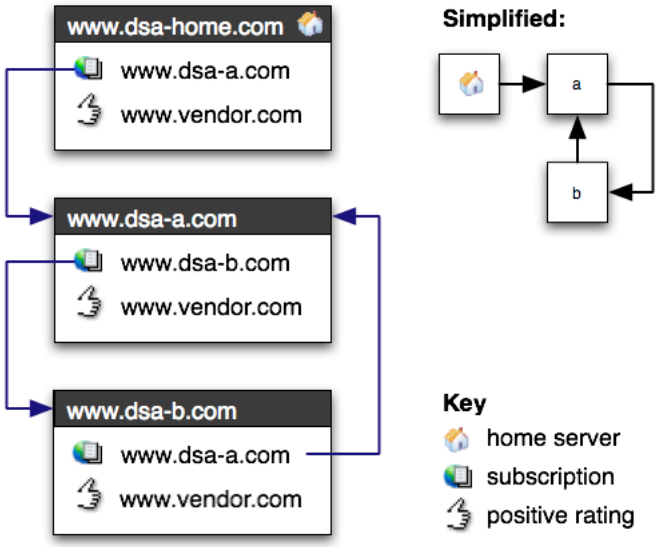
Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.0	✓
www.dsa-a.com www.dsa-home.com	0.5	✓
www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.25	✓

A number of tests were carried out in order to measure the average performance:

Test #	Time Taken (s)	Notes
1	0.82	First browse after browser startup
2	0.87	
3	0.84	
4	0.76	
5	0.86	
Average Time (s):	**0.83**	

Cyclic Subscription Case

In this case a home server is subscribed to a second server, and the second server has a subscription to a third DSA server. The third DSA server also has a subscription to the second, forming a cycle.



In this case the client should receive just three ratings for www.vendor.com, weighted accordingly:

Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.0	✓
www.dsa-a.com www.dsa-home.com	0.5	✓
www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.25	✓

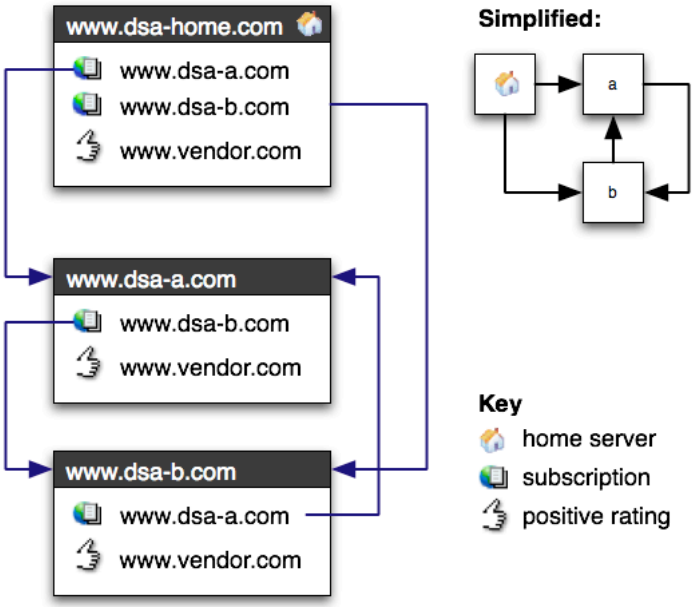
A number of tests were carried out in order to measure the average performance:

Test #	Time Taken (s)	Notes
1	1.39	First browse after browser startup
2	0.79	
3	0.71	
4	0.97	
5	0.76	
Average Time (s):	**0.92**	

Double-Linked Cyclic Subscription Case

In this case a home server is subscribed to a second and third server, and the second server has a subscription to the third DSA server. The third DSA server also has a subscription to the second, forming a cycle.

In most cases, the client should prevent this arrangement from occurring. It will not allow www.dsa-b.com to be subscribed by www.dsa-home.com, because it will detect the presence of www.dsa-b.com in the existing subscription chain from www.dsa-a.com. However, if both www.dsa-a.com and www.dsa-b.com are subscribed before the cycle is created between them, the scenario may be created as follows:

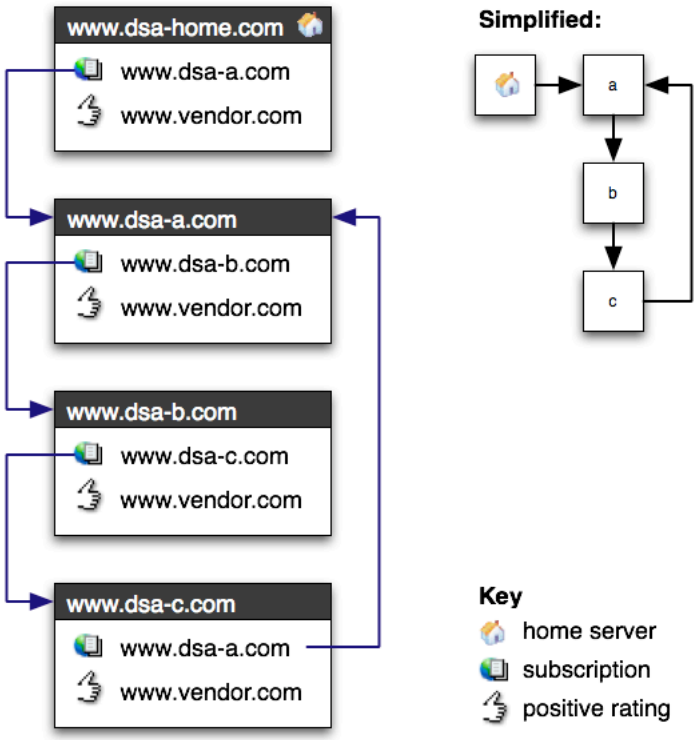


Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.0	✓
www.dsa-a.com www.dsa-home.com	0.5	✓
www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.25	✓
www.dsa-b.com www.dsa-home.com	0.5	✓
www.dsa-a.com www.dsa-b.com www.dsa-home.com	0.25	✓

Test #	Time Taken (s)	Notes
1	1.31	First browse after browser startup
2	1.41	
3	1.24	
4	1.30	
5	1.26	
Average Time (s):	**1.30**	

Larger Cycle Case

In this case a home server is subscribed to a second server, the second server is subscribed to a third server and the third server is subscribed to a fourth. The fourth DSA server also has a subscription to the second, forming a three-node cycle.

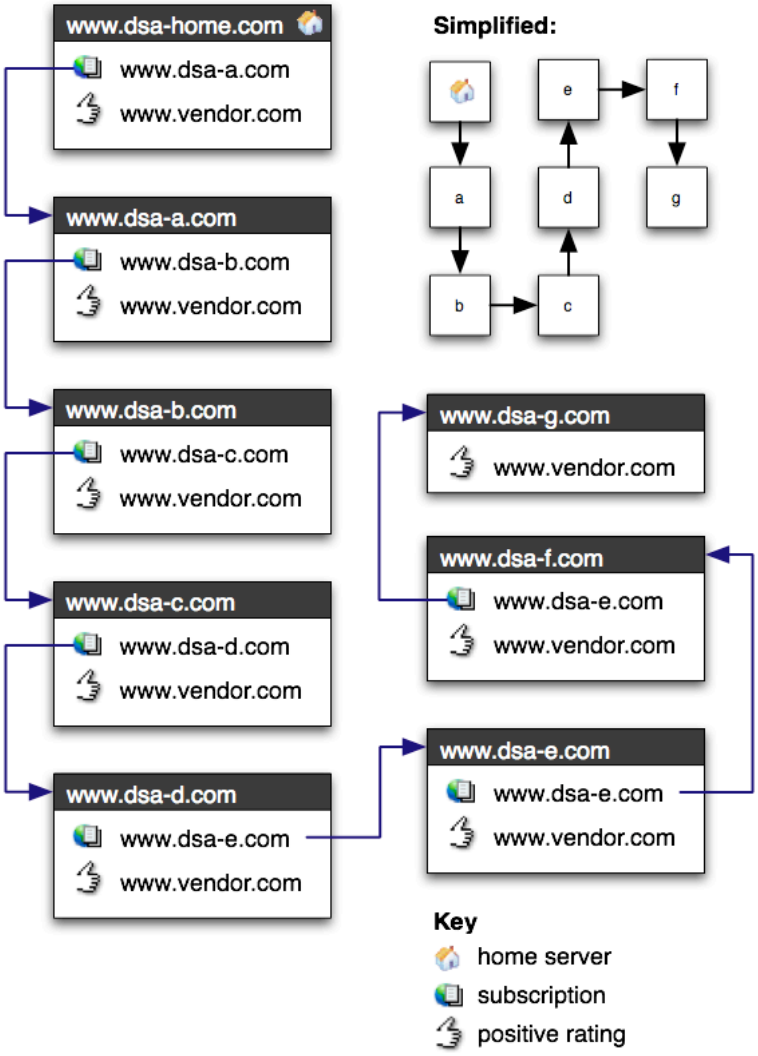


Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.0	
www.dsa-a.com www.dsa-home.com	0.5	
www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.25	
www.dsa-c.com www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.125	

Test #	Time Taken (s)	Notes
1	1.29	First browse after browser startup
2	1.18	
3	1.16	
4	1.11	
5	1.17	
Average Time (s):	**1.18**	

Recursion Limit

In this case a home server is subscribed to a long chain of DSA servers. Since the recursion limit is arbitrarily set to five servers, only five ratings should be aggregated.



Expected serverChain	Expected Weight	Result
www.dsa-home.com	1.000	
www.dsa-a.com www.dsa-home.com	0.500	
www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.250	
www.dsa-c.com www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.125	
www.dsa-d.com www.dsa-c.com www.dsa-b.com www.dsa-a.com www.dsa-home.com	0.063	

Test #	Time Taken (s)	Notes
1	1.52	First browse after browser startup
2	1.40	
3	1.45	
4	1.34	
5	1.37	
Average Time (s):	**1.42**	

Usability Testing

The system has not yet been fully tested for usability. Instead, testing has focussed on the accuracy and performance of the rating aggregation process. Studies into the usability of the system would be necessary before a production release.

Portability

The system has been tested on Microsoft Windows XP and Mac OS X. Further tests should be carried out in other platforms supported by the Firefox web browser:

- Windows 98/ME
- Windows 2000
- Windows NT
- Linux-based operating systems

9 Conclusion

Whilst similar systems exist, the approach taken by this implementation appears to be unique, in that it distributes load, uses social networks to calculate authority and provides relevant information to the user while they browse the Internet. The DSA client combines the advantages of many systems: the simplicity of eBay's positive/negative ratings, the ability to find the reputation of a source of ratings (as in Epinions), and the automated interoperability of agent-based systems (through Web Services).

Despite technological stalling points and changing requirements, the first prototypes were reusable so development continued rapidly. Evolutionary development served the purposes of the project well. The process was unhindered by a reliance on rigid planning techniques.

Testing demonstrated a sound performance of the DSA client and server. Further rigorous usability and load testing would be required before putting the system into production. In particular, the synchronous calls made between DSA servers could prove to introduce a critical bottleneck in complex social networks. To address this problem, an asynchronous-capable Web Services API could be substituted in place of the existing library.

In terms of research contribution, the system does not introduce a groundbreaking new algorithm for reputation analysis. However, of particular interest is its application of social networks and its ability to quickly combine and present traceable reputation data. The system's underlying connection with the Internet, both in terms of infrastructure and data domain means it will scale and become semantically richer with time. It ties in nicely with the concepts of the Semantic Web, as described in the Literature Survey section.

One of the first applications of the system will be with the @lis Technology Net Project. This is a continuously live running network populated by autonomous agents, spanning Europe and Latin America. The DSA system will provide reputation data to the @lis personalised cultural/tourism services. The aim is that software agents and human users can share reputation and the planner agents of the @lis system can use this data when automatically building tourism packages. The system will take advantage of the always-available DSA rating data, and Web Services support built into the DSA server.

10 Bibliography

[1] T. Berners-Lee, J. Hendler and O. Lassila, *The Semantic Web*, *Scientific American*, 2001.

[2] M. Chen and J. P. Singh, *Computing and Using Reputations for Internet Ratings*, ACM (2001).

[3] C. Dudek, *Visual Appeal and the Formation of Trust in E-Commerce Web Sites*, Carleton University, 2003.

[4] J. Golbeck and J. Hendler, *Reputation Network Analysis for Email Filtering*, University of Maryland, 2004.

[5] N. R. Jennings, *Towards a Social Level Characterisation of Socially Responsibble Agents*, Proceedings on Software Engineering, 144 (1997), pp. 11-25.

[6] H. Lei and G. C. Shoja, *A Distributed Trust Model for e-Commerce Applications*, *PANDA Group, Computer Science Department*, University of Victoria, 2004.

[7] M. I. Melnik and J. Alm, *Does a Seller's eCommerce Reputation Matter? Evidence from eBay Auctions*, The Journal of Industrial Economics (2002).

[8] J. S. i. Mir, *Trust and Reputation for Agent Societies*, Consell Superior d'Investigacions Cientifiques (2003).

[9] J. Pujol, *Dynamics of Social Structures in Multi-Agent Systems*, Universitat Politècnica de Catalunya, 2003.

[10] P. Resnick, R. Zeckhauser, J. Swanson and K. Lockwood, *The Value of Reputation on eBay: A Controlled Experiment*, *ESA Conference*, 2002.

[11] W3C, *http://www.w3.org/2004/OWL/*, 2004.

[12] W3C, *http://www.w3.org/RDF/*, W3C, 2004.

[13] G. Zacharia, A. Moukas and P. Maes, *Collaborative Reputation Mechanisms in Electronic Marketplaces*, *32nd Hawaii International Conference on System Sciences*, 1999.

11Appendix A (Source Code)

11.1 DSA Client

11.1.1 contents.rdf

This resource description file describes various properties of the Firefox extension

```
<?xml version="1.0"?>

<RDF:RDF xmlns:RDF="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:chrome="http://www.mozilla.org/rdf/chrome#">

      <RDF:Seq RDF:about="urn:mozilla:package:root">
            <RDF:li RDF:resource="urn:mozilla:package:dsa"/>
      </RDF:Seq>

      <RDF:Seq RDF:about="urn:mozilla:overlays">
            <RDF:li RDF:resource="chrome://browser/content/browser.xul"/>
            <RDF:li RDF:resource="chrome://navigator/content/navigator.xul"/>
      </RDF:Seq>

      <RDF:Seq RDF:about="chrome://browser/content/browser.xul">
            <RDF:li>chrome://dsa/content/dsaOverlay.xul</RDF:li>
      </RDF:Seq>

      <RDF:Seq about="chrome://navigator/content/navigator.xul">
            <RDF:li>chrome://dsa/content/dsaOverlay.xul</RDF:li>
      </RDF:Seq>

      <RDF:Description RDF:about="urn:mozilla:package:dsa"
            chrome:displayName="Don't Shop Alone"
            chrome:author="Chris Beach"
            chrome:authorURL="mailto:chris@chrisbeach.co.uk"
            chrome:name="dsa"
            chrome:extension="true"
            chrome:description="Automatically shares website reviews between friends for
trustworthy online shopping.">
      </RDF:Description>

</RDF:RDF>
```

11.1.2 constants.js

This JavaScript file defines constants used elsewhere in the implementation

```
const DSA_VERSION = "0.2.6";

// UI Files
const XUL_REVIEW = "chrome://dsa/content/review.xul";
const XUL_OPTIONS = "chrome://dsa/content/options.xul";
const XUL_PROTECT_SERVER = "chrome://dsa/content/protectServer.xul";
const XUL_TAKE_OWNERSHIP_OF_SERVER = "chrome://dsa/content/takeOwnershipOfServer.xul";

// URIs, Files, Named Anchors and Query Strings
const URI_FAQ_NO_HOME = "http://www.dontshopalone.co.uk/core/scripts/entryViewer.php?ID=7973";
const URI_FAQ = "http://www.dontshopalone.co.uk/core/scripts/entryViewer.php?ID=7972";
const DSA_SERVER_FILE = "dsa.php";
const QS_SNIFF = "?sniff=true";
const QS_RATINGS_LIST = "?list=true";
const DOMAIN_SEPARATOR = ";";

// Naming of parameters used in WS interaction
const VERSION = 'version';
const DOMAIN = 'domain';
const REVIEW = 'review';
const RATING = 'rating';
const PASSWORD = 'password';
const OLD_PASSWORD = 'oldPassword';
const NEW_PASSWORD = 'newPassword';
const RATING_VALUE = 'ratingValue';
const SERVER_CHAIN = 'serverChain';
const IS_DSA_SERVER = 'isDSAServer';
const IS_PROTECTED = 'isprotected';
const DSA_PREFS_ROOT = "dsa.";

// Rating values
const RATING_NEGATIVE = -1;
const RATING_POSITIVE = 1;

// CSS CLASSES
const CSS_POSITIVE = 'positive';
const CSS_NEGATIVE = 'negative';
const CSS_USER_CHOICE = 'userChoice';
```

11.1.3 dsaOverlay.css

This is a cascading style sheet document that sets styles for the toolbar interface.

```
.negative
{
      background-color: #fee;
      color: #633
}
.positive
{
      background-color: #efe;
      color: #363
}

.dsaSeparator
{
  margin: 2px; width: 2px; height: 16px;
  border-left: 1px dotted #999;
  border-right: 1px dotted #ccc
}

.userChoice { font-weight: bold }

.odd { background-color: #cce }
.even {      background-color: #dde }

#ratingValue[value="5"] { list-style-image: url("chrome://dsa/content/skin/3ThumbsUp.png"); }
#ratingValue[value="4"] { list-style-image: url("chrome://dsa/content/skin/2ThumbsUp.png"); }
#ratingValue[value="3"] { list-style-image: url("chrome://dsa/content/skin/thumbsUp.png"); }
#ratingValue[value="2"] { list-style-image: url("chrome://dsa/content/skin/thumbsDown.png"); }
#ratingValue[value="1"] { list-style-image: url("chrome://dsa/content/skin/2ThumbsDown.png"); }
#ratingValue[value="0"] { list-style-image: url("chrome://dsa/content/skin/3ThumbsDown.png"); }
```

11.1.4 dsaOverlay.js

This JavaScript file contains the program logic for the toolbar

```
/**
  * Don't Shop Alone (DSA) Client
  * Chris Beach 2005
  */

// Global variables
var gIsInitialised = false;
var gObjURIHome = null;
var gIsHomeOwner = false;
var gPassword = "";
var gSiteState = null;
var gPrefs = null;

// DOM UI Objects
var gObjRatingValueButton;
var gObjRatingValue;

var gObjRatingButtons;
var gObjPositiveRating;
var gObjNegativeRating;

var gObjReviewsMenuButton;
var gObjReviewsMenu;

var gObjMenuitemListRatings;
var gObjMenuitemWarnAgainst;
var gObjMenuitemTakeOwnership;

var gObjMyHomeServer;
var gObjInfoNoHome;
var gObjDSAServerFound;
var gObjSubscribe;
var gObjSubscribed;

var gObjWaitImage;

// Get IO Server
var gIOService = Components.classes["@mozilla.org/network/io-service;1"].getService();
gIOService = gIOService.QueryInterface(Components.interfaces.nsIIOService);

// Initialise the client when the browser loads
document.addEventListener( "load", initialiseClient, true );

/**
 * Called after the browser has finished loading
 */
function initialiseClient()
{
      if ( gIsInitialised ) return false;
      gIsInitialised = true;

      // Set global references to UI DOM objects
      gObjRatingValueButton = document.getElementById( "ratingValueButton" );
      gObjRatingValue = document.getElementById( "ratingValue" );

   gObjRatingButtons = document.getElementById( "ratingButtons" );
      gObjPositiveRating = document.getElementById( "positiveRating" );
      gObjNegativeRating = document.getElementById( "negativeRating" );

      gObjReviewsMenuButton = document.getElementById( "reviewsMenuButton" );
      gObjReviewsMenu = document.getElementById( "reviewsMenu" );

      gObjMenuitemListRatings = document.getElementById( "menuitemListRatings" );
      gObjMenuitemWarnAgainst = document.getElementById( "menuitemWarnAgainst" );
      gObjMenuitemTakeOwnership = document.getElementById( "menuitemTakeOwnership" );

      gObjMyHomeServer = document.getElementById( "myHomeServer" );
```

```
	gObjInfoNoHome = document.getElementById( "noDSAHome" );
	gObjDSAServerFound = document.getElementById( "dsaServerFound" );
	gObjSubscribe = document.getElementById( "subscribe" );
	gObjSubscribed = document.getElementById( "subscribed" );

	gObjWaitImage = document.getElementById( "waitImage" );

	dump( "Initialising DSA Client Version " + DSA_VERSION + "\n\n" );

	// Apply listener function to webpage load events
	document.getElementById( "content" ).addProgressListener( dsaListener,
		Components.interfaces.nsIWebProgress.NOTIFY_STATE_DOCUMENT );

	// Apply listener function to webpage unload events
	document.getElementById( "content" ).addEventListener( "unload", pageUnloadHandler, true );

	// Get preferences from Mozilla preferences table
	gPrefs = getPrefs();

	try	{ gPassword = gPrefs.getCharPref( PASSWORD ); }
	catch( e ) { gPrefs.setCharPref( PASSWORD, "" ); }

	try	{ gIsHomeOwner = gPrefs.getBoolPref( "isHomeOwner" ); }
	catch( e ) { gPrefs.setBoolPref( "isHomeOwner", false ); }

	// Get the URI of the DSA home server (if exists)
	var objURI = null;
	try	{ objURI = gIOService.newURI( gPrefs.getCharPref( "uri.home" ), null, null ); }
	catch( e ) { gPrefs.setCharPref( "uri.home", "" ); }
	setHomeURI( objURI );
}

/**
 * Called on the loading of a new page
 */
function pageLoadHandler( objURI )
{
	if ( !objURI || !objURI.host )
		return;

	dump( "\npageLoadHandler() currentURI = [" + objURI.host + objURI.path + "]\n" );

	if ( !gSiteState || objURI.host != gSiteState.objURI.host )
	{
		dump( 'Browsed to new domain\n' );

		// Instantiate a whole new site state
		gSiteState = new SiteState( objURI );
		updateUI();

		// We have browsed to a new domain
		// Check if the current site is the user's DSA home server
		if ( !gSiteState.isMyHome() )
		{
			dump( 'Site is not my home, getting ratings\n' );
			//	Initiate a request for ratings
			//	The callback function will display the rating
			initGetRating( gSiteState.objURI );
		}
		else
		{
			dump( 'Site is my home, not getting new ratings\n' );
			gSiteState.gotRatings = true;
		}
		// Initiates sniffer function to detect DSA server
		//		perform domain and path-level searches
		dump( "pageLoadHandler()::initSniffServer( [uri], true )\n" );
		initSniffServer( gSiteState.objURI, true );
	}
	else if ( objURI.host == gSiteState.objURI.host && objURI.path == gSiteState.objURI.path )
	{
		// No change of domain or path
```

```
                dump( 'No change of URI\n' );
                return;
        }
        else
        {
                dump( 'Browsed to new path\n' );
                gSiteState.objURI = objURI;
                updateUI();

                // Initiates sniffer function to detect DSA server
                //             perform just path-level search
                dump( "pageLoadHandler()::initSniffServer( [uri], false )\n" );
                initSniffServer( gSiteState.objURI, false );
        }
}

/**
 * Sets the home server from which to source ratings
 * Takes a URI of the form http://www.dontshopalone.co.uk/
 */
function setHomeURI( objURI )
{
        var strURI = "";

        if ( objURI )
        {
                dump( "setHomeURI( " + objURI.spec + " )\n" );
                strURI = objURI.spec;
        }
        else
        {
                dump( "setHomeURI( null )\n" );
                gPrefs.setCharPref( PASSWORD, "" );
                gIsHomeOwner = false
                gPrefs.setBoolPref( "isHomeOwner", gIsHomeOwner );
        }
        gObjURIHome = objURI;
        gPrefs.setCharPref( "uri.home", strURI )
        // Refreshes relevant UI elements
        updateUI();
}

/**
 * Opens the help system
 */
function openDSAHelp()
{
        window._content.document.location.href = URI_HELP;
}

/**
 * Called when a webpage is closed or navigated away from
 */
function pageUnloadHandler( e )
{
        dump( 'pageUnloadHandler\n' );
}

/**
 * Takes an array of rating objects and updates the toolbar
 * to display them (adds any reviews to review menu)
 */
function displayRatings( ratingArray )
{
        if ( !ratingArray )
                return;

      //dump( 'displayRatings( array.length=' + ratingArray.length + ' )' );
        if ( gObjURIHome && gSiteState.objServerURI &&
                gObjURIHome.spec == gSiteState.objServerURI.spec )
                // User is browsing their own DSA server - don't show ratings
                return;
```

```
  var ratingSum = 0;
 var reviewCount = 0;

  // Flush reviews from menu
  for( var i = gObjReviewsMenu.childNodes.length - 1; i >= 0; i-- )
  {
    gObjReviewsMenu.removeChild( gObjReviewsMenu.childNodes.item( i ) );
 }

  var consideredServers = new Array();

  if ( ratingArray.length > 0 )
  {
        var positiveCount = 0;
        var negativeCount = 0;
        var positiveWeight = 0.0;
        var negativeWeight = 0.0;
        // Populate menu and average the ratings
        for( var i = 0; i < ratingArray.length; i++ )
        {
                // Check to see if we've already seen a rating from this server
                var alreadyConsidered = false;
                for( var j = 0; j < consideredServers.length; j++ )
                {
                        if ( consideredServers[j] == ratingArray[i].getAuthorURI() )
                        {
                                alreadyConsidered = true;
                                continue;
                        }
                }
                if ( !alreadyConsidered )
                        consideredServers[ consideredServers.length ] =
ratingArray[i].getAuthorURI();

                var cssClass;
                ratingSum += ratingArray[i].ratingValue;

                if ( ratingArray[i].ratingValue == RATING_POSITIVE )
                {
                        positiveCount++;
                        positiveWeight += ratingArray[i].getWeightedValue();
                }
                else if ( ratingArray[i].ratingValue == RATING_NEGATIVE )
                {
                        negativeCount++;
                        negativeWeight -= ratingArray[i].getWeightedValue();
                }

                if ( ratingArray[i].review && !alreadyConsidered )
                {
                        // Sets CSS (stylesheet) class - modifies colour of menuitem
                        if ( ratingArray[i].ratingValue == RATING_POSITIVE )
                                cssClass = CSS_POSITIVE;
                        else
                                cssClass = CSS_NEGATIVE;

                        var newMenuItem = document.createElement( "menuitem" );
                        newMenuItem.setAttribute( "class", cssClass );
                        newMenuItem.setAttribute( "tooltiptext", ratingArray[i].serverChain );

                        newMenuItem.setAttribute( "label",
                                ratingArray[i].getAuthor() + ": \"" +
                                ratingArray[i].review + "\"" );
                        gObjReviewsMenu.appendChild( newMenuItem );
                        reviewCount++;
                }
        }
        if ( reviewCount > 0 )
        {
                if ( reviewCount == 1 )
                        gObjReviewsMenuButton.label = reviewCount + " review";
```

```
                else if ( reviewCount > 1 )
                        gObjReviewsMenuButton.label = reviewCount + " reviews";
        }
        var ratingValue;

        // No negative votes
        if ( negativeCount == 0 && positiveWeight > 5 )
                ratingValue = 5;
        else if ( negativeCount == 0 && positiveWeight > 1 )
                ratingValue = 4;
        else if ( negativeCount == 0 )
                ratingValue = 3;

        // One or more negative vote(s)
        else if ( positiveWeight / negativeWeight > 50 )
                ratingValue = 5;
        else if ( positiveWeight / negativeWeight > 20 )
                ratingValue = 4;
        else if ( positiveWeight / negativeWeight > 10 )
                ratingValue = 3;
        else if ( positiveWeight / negativeWeight > 5 )
                ratingValue = 2;
        else if ( positiveWeight / negativeWeight > 0.2 )
                ratingValue = 1;
        else if ( negativeCount > 2 )
                ratingValue = 0;
        else
                ratingValue = 1;

        gObjRatingValueButton.setAttribute( "tooltiptext",
                positiveCount + " positive vote(s), " +
                negativeCount + " negative vote(s)" );

        dump( "positive weighting: " + positiveWeight +
                ", negative weighting: " + negativeWeight + "\n" );

        // Change the arbitrary "value" attribute added to the DOM object
        //   the stylesheet will then change the image accordingly
        gObjRatingValue.setAttribute( "value", ratingValue );
    }
}

/**
 * Updates all elements of the user interface depending on the site state
 */
function updateUI()
{
    if ( !gSiteState )
    {
        dump( 'No SiteState\n' );
        return false;
    }
    var isDisplayable = gSiteState.isDisplayable();

    // Show/hide the animated hourglass icon
    gObjWaitImage.hidden = isDisplayable;

    // Disable/enable controls is the site state is not displayable
    gObjPositiveRating.disabled = !isDisplayable;
    gObjNegativeRating.disabled = !isDisplayable;
    gObjReviewsMenuButton.disabled = !isDisplayable;
    gObjRatingValue.disabled = !isDisplayable;
    gObjMyHomeServer.disabled = !isDisplayable;
    gObjDSAServerFound.disabled = !isDisplayable;
    gObjSubscribe.disabled = !isDisplayable;
    gObjSubscribed.disabled = !isDisplayable;

    if( !isDisplayable )
    {
        dump( 'SiteState not displayable: gotRatings=' + gSiteState.gotRatings +
                ', gotSniff=' + gSiteState.gotSniff + '\n' );
        return false;
```

```
	}

	if ( gSiteState.isUnprotectedServer )
	{
		// Found a virginal unprotected server
		foundUnprotectedServer();
		return;
	}

	displayRatings( gSiteState.ratings );

	if ( !gSiteState.usersRating )
	{
		// User has not rated the current site
		gObjPositiveRating.setAttribute( "class", "" );
		gObjNegativeRating.setAttribute( "class", "" );
	}
	else if ( gSiteState.usersRating.ratingValue == RATING_POSITIVE )
	{
		gObjPositiveRating.setAttribute( "class", CSS_USER_CHOICE );
		gObjNegativeRating.setAttribute( "class", "" );
	}
	else if ( gSiteState.usersRating.ratingValue == RATING_NEGATIVE )
	{
		gObjPositiveRating.setAttribute( "class", "" );
		gObjNegativeRating.setAttribute( "class", CSS_USER_CHOICE );
	}

	if ( gSiteState.isDSAServer() )
	{
		// User has browsed to a DSA server
		gObjSubscribe.setAttribute( "oncommand",
			"getRatingsFrom( \"" + gSiteState.objServerURI.spec + "\" );" );

		gObjMenuitemTakeOwnership.setAttribute( "oncommand",
			"takeOwnership( \"" + gSiteState.objServerURI.spec + "\" );" );

		// Set the logic for the 'list ratings' button
		gObjMenuitemListRatings.setAttribute( "oncommand",
			"navigateToURI( \"" + gSiteState.objServerURI.spec + DSA_SERVER_FILE +
QS_RATINGS_LIST + "\" );" );
	}

	dump( "State: isMyHome()=[" + gSiteState.isMyHome() +
		"], isDSAServer()=[" + gSiteState.isDSAServer() +
		"], isSubscribed=[" + gSiteState.isSubscribed + "]\n" );

	gObjRatingButtons.hidden = gSiteState.isDSAServer() || gSiteState.isMyHome() || (
gObjURIHome == null ) || !gIsHomeOwner;
	gObjMyHomeServer.hidden = !gSiteState.isMyHome();
	gObjDSAServerFound.hidden = !gSiteState.isDSAServer();
	gObjSubscribe.hidden = !gSiteState.isDSAServer()
	gObjMenuitemTakeOwnership.hidden = gSiteState.isMyHome();
	gObjMenuitemWarnAgainst.hidden = !gSiteState.isDSAServer() ||
		!gIsHomeOwner ||
		gSiteState.isMyHome();
	gObjInfoNoHome.hidden = gSiteState.isDSAServer() || ( gObjURIHome != null );
	gObjSubscribed.hidden = !gSiteState.isDSAServer() ||
		!gSiteState.isSubscribed ||
		gSiteState.isMyHome();
	gObjSubscribe.hidden = !gSiteState.isDSAServer() ||
		gSiteState.isSubscribed ||
		gSiteState.isMyHome();

	var showRatingsAndReviews = ( gSiteState.ratings && gSiteState.ratings.length > 0 ) &&
		!gSiteState.isMyHome();

	gObjRatingValue.hidden = !showRatingsAndReviews;
	gObjReviewsMenuButton.hidden = !showRatingsAndReviews || gSiteState.reviewCount == 0;
}
```

11.1.5 dsaOverlay.xul

This XML file describes the toolbar user interface

```
<?xml version="1.0"?>
<?xml-stylesheet href="dsaOverlay.css" type="text/css"?>

<overlay id="dsaOverlay"
         xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul">

// Imports JavaScript
<script type="application/x-javascript" src="constants.js" />
<script type="application/x-javascript" src="util.js" />
<script type="application/x-javascript" src="Rating.js" />
<script type="application/x-javascript" src="SiteState.js" />
<script type="application/x-javascript" src="io.js" />
<script type="application/x-javascript" src="dsaOverlay.js" />

// DSA Toolbar
<toolbox id="navigator-toolbox">
<toolbar id="dsaToolbar" class="chromeclass-toolbar"
      toolbarname="Don't Shop Alone Toolbar">

      // Options Menu
      <toolbarbutton type="menu" id="optionsMenuButton" label=""
              showlabel="" container="true" tooltiptext="Set options"
              image="chrome://dsa/content/skin/dsaLogo.png">
              <menupopup id="optionsMenu">
                      <!--<menuitem label="Setup" oncommand="openDSAOptions();" />-->
                      <menuitem label="Help" oncommand="navigateToURI( URI_FAQ );" />
              </menupopup>
      </toolbarbutton>

      // Sniffed DSA Server
      <hbox id="dsaServerFound" hidden="true">
              <toolbarbutton type="menu" container="true"
                      tooltiptext=" This site provides ratings"
                      label=" This site provides ratings">
                      <menupopup id="serverSniffMenu">
                              <menuitem id="menuitemListRatings"
                                      label="View ratings from this DSA server" />
                              <menuitem id="menuitemTakeOwnership"
                                      label="This is my own server (requires password)" />
                              <menuitem id="menuitemWarnAgainst" oncommand="saveRatingValue( -1 );"
                                      tooltiptext="Warn against this ratings source"
                                      label="The ratings provided are unrepresentative"
                                      src="chrome://dsa/content/skin/thumbsDown.png" />
                      </menupopup>
              </toolbarbutton>

              // Subscribe
              <toolbarbutton id="subscribe" image="chrome://dsa/content/skin/subscribe.png"
                      label=" Get ratings from here" />

              // Subscribed
              <hbox id="subscribed" hidden="true">
                      <vbox class="dsaSeparator" />
                      <toolbarbutton image="chrome://dsa/content/skin/info.png"
                              label=" I receive ratings from here">
                      </toolbarbutton>
              </hbox>
      </hbox>

      // Rating buttons
      <hbox id="ratingButtons" hidden="true">
              <vbox class="dsaSeparator" />
              <toolbarbutton id="positiveRating" onclick="saveRatingValue( 1 );"
                      tooltiptext="Recommend this site"
                      showlabel=" I recommend this site"
                      label=" I recommend this site" firstlabel=" I recommend this site"
                      image="chrome://dsa/content/skin/thumbsUp.png" />
```

```
            <toolbarbutton id="negativeRating" onclick="saveRatingValue( -1 );"
                    tooltiptext="Warn against this site"
                    showlabel=" I received bad service from this site"
                    label=" I received bad service from this site"
                    firstlabel=" I received bad service from this site"
                    image="chrome://dsa/content/skin/thumbsDown.png" />
    </hbox>

    // Rating Text
    <hbox id="ratingValue" hidden="true">
            <vbox class="dsaSeparator" />
            <toolbarbutton id="ratingValueButton" onclick="" disabled="true"
                    tooltiptext="Rating Value" showlabel="Rating: "
                    label="Rating: " firstlabel="Rating " dir="rtl" />
    </hbox>

    // Review Menu
    <toolbarbutton type="menu" hidden="true" id="reviewsMenuButton"
            container="true" tooltiptext="Read top reviews">
            <menupopup id="reviewsMenu">
            </menupopup>
    </toolbarbutton>

    // Home server
    <toolbarbutton id="myHomeServer"
            label=" (This is my home server)" hidden="true">
    </toolbarbutton>

    // No Home
    <hbox id="noDSAHome" hidden="true">
            <vbox class="dsaSeparator" />
            <toolbarbutton type="menu" id="infoMenuButton"
                    container="true" image="chrome://dsa/content/skin/info.png"
                    label=" No ratings source">
                    <menupopup id="noHomeMenu">
                            <menuitem label="What does this mean?"
                                    oncommand="navigateToURI( URI_FAQ_NO_HOME );" />
                    </menupopup>
            </toolbarbutton>
    </hbox>

    <spring flex="1"/>

    // Animated hourglass icon
    <toolbarbutton id="waitImage" image="chrome://dsa/content/skin/hourglass.gif" />

</toolbar>
</toolbox>

</overlay>
```

11.1.6 io.js

This JavaScript file contains methods that communicate with external hosts

```
/**
 * Initialises Web Services call to save a rating
 */
function initSaveRating( rating )
{
	dump( "initSaveRating()\n" );

	// Change state information and update the UI to show the hourglass
	gSiteState.gotRatings = false;
	updateUI();

	if ( gObjURIHome == null )
	{
		alert( "You have not configured a DontShopAlone server to save ratings to." );
		return false;
	}

	if ( rating.uri == gObjURIHome.spec && rating.ratingValue == RATING_NEGATIVE &&
		confirm( "You will no longer receive ratings from this server. Continue?" ) )
	{
		dump( 'User has deemed his home server untrustworthy\n' );
		setHomeURI( null );
		return false;
	}

	if( rating.uri != gObjURIHome.spec && rating.ratingValue == RATING_NEGATIVE )
	{
		// Negative rating - Prompt the user for a review
		window.openDialog( XUL_REVIEW,
			"Review", "chrome,modal,dialog,dependent,centerscreen", rating );
		if ( rating.review == "" )
		{
			dump( "Rating is negative and no review given; not saving the rating\n" );
			return false;
		}
	}
	var call = new SOAPCall();
	call.transportURI = gObjURIHome.spec + DSA_SERVER_FILE;
	call.actionURI = gObjURIHome.spec + DSA_SERVER_FILE + "/addRating";

	dump( 'Saving rating: [' + rating + ']\n' );

	var params = new Array(
		new SOAPParameter( rating, RATING ),
		new SOAPParameter( gPassword, PASSWORD )
	);
	call.encode( 0, "addRating", "urn:RatingService", 0, null, params.length, params );
	var callRequest = call.asyncInvoke( function( callResponse, call, status )
		{
			if ( callResponse.fault )
			{
				alert( callResponse.fault.faultString );
			}
			// Load new rating
			dump( "Getting new rating: initGetRating on [" + rating.uri + "]\n" );
			initGetRating( gIOService.newURI( rating.uri, null, null) );
		} );
	if ( callRequest.fault )
	{
		alert( callRequest.fault.faultString );
	}
	return true;
}

/**
 * Initialise an XMLHTTPRequest to find the DSA server at the currently
 * browsed site
```

```
 */
function initSniffServer( objURI, isDomainLevelScan )
{
	dump( "initSniffServer( " + objURI.host + ", " + isDomainLevelScan + ")\n" );

	var uriHasPath = hasPath( objURI );
	var objScanURI = null;

	if ( isDomainLevelScan )
		objScanURI = truncateURIPathAndFile( objURI );
	else
		objScanURI = truncateURIFile( objURI );

	var strScanURI = objScanURI.spec + DSA_SERVER_FILE + QS_SNIFF;

	// Create a new XMLHTTPRequest
	var reqSniff = new XMLHttpRequest();

	// Set the call-back function for the request
	reqSniff.onreadystatechange = function()
	{
		if ( reqSniff.readyState == 4 )
		{
			if ( reqSniff.status == 200 )
			{
				// DSA file was FOUND
				gSiteState.gotSniff = true;
				try
				{
					dump( "sniffDSAServer callback: readyState = [" +
reqSniff.readyState +
						"], status=[" + reqSniff.status + "]\n" );
					if ( !reqSniff.responseXML ) return false;

					var responseXML = reqSniff.responseXML.documentElement;
					// Extract attributes from the XML response
					var dsaVersion = responseXML.attributes[ VERSION ].value;
					var isProtected = responseXML.attributes[ IS_PROTECTED
].value;
					gSiteState.objServerURI = objScanURI;
					gSiteState.isUnprotectedServer = ( isProtected == 'false' )
				}
				catch ( e ) { dump( "Error in sniffDSACallback - [" + e + "]\n" ); }
				updateUI();
			}
			else if ( reqSniff.status == 404 && isDomainLevelScan && uriHasPath )
			{
				// Server not found. Try again at the path level
				dump( "No DSA server found on domain-level, performing path-level
scan\n" );
				// No server found at domain level -- attempt at the file level
				initSniffServer( objURI, false )
			}
			else
			{
				gSiteState.gotSniff = true;
				updateUI();
			}
		}
	}
	dump( "sniffing for ( " + strScanURI + ")\n" );
	reqSniff.open( "GET", strScanURI, true )
	reqSniff.send( null );
}

function initGetRating( objURI )
{
	dump( "initGetRating( " + objURI.host + " )\n" );

	if ( !objURI.host ) return false;
	if ( gObjURIHome == null ) return false;
```

```
	gSiteState.initGetRatingsTimestamp = new Date();

	var call = new SOAPCall();
	dump( "transportURI = [" + gObjURIHome.spec + DSA_SERVER_FILE + "]\n" );
	call.transportURI = gObjURIHome.spec + DSA_SERVER_FILE;
	call.actionURI = gObjURIHome.spec + DSA_SERVER_FILE + "/getRatings";

	var params = new Array();
	params[0] = new SOAPParameter( objURI.host, DOMAIN );
	params[1] = new SOAPParameter( "", SERVER_CHAIN );
	call.encode( 0, "getRatings", "urn:RatingService", 0, null, params.length, params );
	var callRequest = call.asyncInvoke( getRatingResponse );
	if ( callRequest.fault )
	{
		alert( callRequest.fault.faultString );
	}
	return true;
}

/**
 * Callback function from initGetRating()
 */
function getRatingResponse( callResponse, call, status )
{
	gSiteState.gotRatings = true;
	dump( "getRatingResponse(...)\n" );
	if ( callResponse.fault )
	{
		dump( callResponse.fault.faultString );
		return false;
	}
	var params = callResponse.getParameters( false, {} );
	if ( !params )
	{
		dump( "No return data\n" );
		updateUI();
		return false;
	}
	var ratingArray = new Array();
	try
	{
		var domRatingArray = params[0].element.getElementsByTagName( "item" );
		for( var i = 0; i < domRatingArray.length; i++ )
		{
			var rating = new Rating();
			if ( domRatingArray[i].getElementsByTagName( RATING_VALUE )[0].firstChild )
				rating.ratingValue = parseInt(
					domRatingArray[i].getElementsByTagName( RATING_VALUE
)[0].firstChild.nodeValue );
			else
				continue;
			if ( domRatingArray[i].getElementsByTagName( REVIEW )[0].firstChild )
				rating.review = domRatingArray[i].getElementsByTagName( REVIEW
)[0].firstChild.nodeValue;
			if ( domRatingArray[i].getElementsByTagName( SERVER_CHAIN )[0].firstChild )
				rating.serverChain = domRatingArray[i].getElementsByTagName(
SERVER_CHAIN )[0].firstChild.nodeValue;
			if ( domRatingArray[i].getElementsByTagName( IS_DSA_SERVER )[0].firstChild )
			{
				dump( 'getRatingResponse rating.isDSAServer=[' +
domRatingArray[i].getElementsByTagName( IS_DSA_SERVER )[0].firstChild.nodeValue + ']\n' );
				rating.isDSAServer = ( domRatingArray[i].getElementsByTagName(
IS_DSA_SERVER )[0].firstChild.nodeValue == 'true' );
			}

			dump( "Got rating: [" + rating.ratingValue + "], review: [" + rating.review +
"], weighted: [" + rating.getWeightedValue() + "], serverChain: [" + rating.serverChain + "]\n" );
			ratingArray[i] = rating;
		}
	}
	catch( e ) { dump( 'Exception parsing ratings return [ ' + e + ' ] ' + e.stack + '\n' ) }
	gSiteState.setRatings( ratingArray );
```

```
	try
	{
		var timeNow = new Date();
		var msTaken = timeNow.getTime() - gSiteState.initGetRatingsTimestamp.getTime();
		dump( "Time taken to get ratings: " + ( msTaken / 1000 ) + "s\n" );
	}
	catch( e ) { dump( e + ' ' + e.stack ); }

	updateUI();
}

/**
 * Wrapper function for initSaveRating - takes only a rating value (numeric)
 */
function saveRatingValue( ratingValue )
{
	dump( "saveRatingValue( " + ratingValue + " )\n" );
	var rating;

	rating = new Rating();

	if ( gSiteState.usersRating &&
		gSiteState.usersRating.ratingValue == ratingValue )
	{
		// Give the user back their previous review
		rating.review = gSiteState.usersRating.review;
	}
	rating.domain = gSiteState.objURI.host;
	rating.uri = truncateURIFile( gSiteState.objURI ).spec;
	rating.ratingValue = ratingValue;
	initSaveRating( rating );
}

/**
 * Prompts user for password for the currently-browsed DSA server
 * If password correct, set the DSA server to be the user's DSA home
 */
function takeOwnership( strURI )
{
	dump( "takeOwnership( " + strURI + " )\n" );
	var objURI = gIOService.newURI( strURI, null, null);
	var password = new String();
	window.openDialog( XUL_TAKE_OWNERSHIP_OF_SERVER, "Take Ownership of Server",
		"chrome,modal,dialog,centerscreen,dependent,titlebar", password );
	dump( "password=[" + password.value + "]" );
	if ( !password.value ) return false;

	// Set the client preference entry
	gPrefs.setCharPref( PASSWORD, password.value );

	// Check the server password by attempting to change it (to the same value)
	changePersonalServerPassword(
		password.value,
		password.value,
		function( callResponse, call, status )
		{
			if ( callResponse.fault )
			{
				// Should detect the fault code rather than assume it is this:
				alert( "Password incorrect" );
				//alert( callResponse.fault.faultString );
			}
			else
			{
				alert( objURI.host + " is now set as your DontShopAlone home server"
);
				gIsHomeOwner = true;
				gPrefs.setBoolPref( "isHomeOwner", gIsHomeOwner );
				gPassword = password.value;
				gPrefs.setCharPref( PASSWORD, gPassword );
				setHomeURI( objURI );
```

```
                        }
                }
        );
}

/**
 * User has browsed to a virgin, unprotected DSA server
 * Prompt the user for a password to initialise the server
 */
function foundUnprotectedServer()
{
        var password = new String();
        window.openDialog( XUL_PROTECT_SERVER, "Protect Server",
                "chrome,modal,dialog,centerscreen,dependent,titlebar", password );
        dump( "password=[" + password.value + "]" );
        if ( !password.value ) return false;

        // Set the client preference entry
        gPrefs.setCharPref( PASSWORD, password.value );

        // Set the server preference entry
        changePersonalServerPassword(
                "",
                password.value,
                function( callResponse, call, status )
                {
                        gSiteState.isUnprotectedServer = false;
                        if ( callResponse.fault )
                        {
                                alert( callResponse.fault.faultString );
                        }
                        else
                        {
                                alert( "Password set" );
                                gIsHomeOwner = true;
                                gPrefs.setBoolPref( "isHomeOwner", gIsHomeOwner );
                                gPassword = password.value;
                                gPrefs.setCharPref( PASSWORD, gPassword );
                                setHomeURI( gSiteState.objServerURI );
                        }
                }
        );
}

/**
 * Initialise the Web Services call to change the password of a DSA server
 * Use the passed callback function to receive the result
 */
function changePersonalServerPassword( oldPassword, newPassword, callback )
{
        if( !gSiteState.objServerURI )
        {
                dump( "Error: changePersonalServerPassword() - no detected DSA server\n" );
                return false;
        }
        var call = new SOAPCall();
        dump( 'transportURI = ' + gSiteState.objServerURI.spec + DSA_SERVER_FILE + "\n" );
        call.transportURI = gSiteState.objServerURI.spec + DSA_SERVER_FILE;
        call.actionURI = gSiteState.objServerURI.spec + DSA_SERVER_FILE + "/changePassword";
        var params = new Array();
        params[0] = new SOAPParameter( oldPassword, OLD_PASSWORD );
        params[1] = new SOAPParameter( newPassword, NEW_PASSWORD );
        call.encode( 0, "changePassword", "urn:RatingService", 0, null, params.length, params );
        var callRequest = call.asyncInvoke( callback );
        if ( callRequest.fault )
        {
                alert( callRequest.fault.faultString );
        }
}

/**
 * If the user is a homeowner, subscribe to ratings from this DSA server
```

```
 *           (add a positive rating for it)
 * If the user is not a homeowner, set this DSA server to be the user's home
 */
function getRatingsFrom( strURI )
{
      var objURI = gIOService.newURI( strURI, null, null)

      if ( gObjURIHome != null && gIsHomeOwner )
      {
            // User owns a server, subscribe to this source
            var rating = new Rating();
            rating.ratingValue = RATING_POSITIVE;
            rating.domain = objURI.host;
            rating.uri = objURI.spec;
            rating.isDSAServer = true;
            initSaveRating( rating )
      }
      else
      {
            // Set this to be the new ratings source
            var objURIServer = truncateURIFile( objURI );
            setHomeURI( objURIServer );
            // Required in order to update site state
            initGetRating( objURIServer );
      }
}
```

11.1.7 protectServer.xul

This XML file describes the dialog presented when the user browses to an unprotected server.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<dialog id="dsaProtectServerDialog" title="Don't Shop Alone - Protect Server"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  buttons="accept,cancel"
  ondialogaccept="return onAccept();"
  ondialogcancel="return onCancel();">

<script type="application/x-javascript">
<![CDATA[

function onAccept()
{
      // The rating is passed to this dialog as an argument
      var password = window.arguments[0];
      password.value = document.getElementById( "txtPassword" ).value;

      if ( !password.value )
      {
            alert( "Please enter a password" );
            return false;
      }
      else
            return true;
}

function onCancel()
{
      return confirm( "Are you sure you want cancel setting a password?" );
}

 ]]>
</script>

<vbox>
      <label control="txtReview" value="Please choose a password for your DontShopAlone server:"
/>
      <textbox id="txtPassword" maxlength="25" />
      <description style='max-width: 400px'>
```

```
            This is mandatory. The password will be used by the server to ensure that noone
            else can save ratings onto it.
      </description>
</vbox>

</dialog>
```

11.1.8 Rating.js

This JavaScript file describes a Rating object

```
/**
 * Rating object encapsulates a rating on a domain
 */
function Rating()
{
	this.domain = "";
	this.uri = "";
	this.ratingValue = 0;
	this.review = "";
	this.serverChain = "";
	this.isDSAServer = false;
	this.toString = ratingToString;
	this.getAuthor = ratingGetAuthor;
	this.getWeightedValue = ratingGetWeightedValue;
	this.getAuthorURI = ratingGetAuthorURI;
}

/**
 * Returns a string representation of the rating
 */
function ratingToString()
{
	var str = "";
	str += "domain=" + URLEncode( this.domain );
	str += "&uri=" + URLEncode( this.uri );
	str += "&ratingValue=" + this.ratingValue;
	str += "&review=" + URLEncode( this.review );
	str += "&isDSAServer=" + ( this.isDSAServer ? "true" : "false" );
	str += "&serverChain=" + URLEncode( this.serverChain );
	return str;
}

/**
 * Returns the ratingValue, halved for each hop it has travelled
 */
function ratingGetWeightedValue()
{
	var hopCount = this.serverChain.split( DOMAIN_SEPARATOR ).length - 1;
	return this.ratingValue / Math.pow( 2, hopCount );
}

function ratingGetAuthorURI()
{
	if ( !this.serverChain )
		return "";
	else
		return this.serverChain.split( DOMAIN_SEPARATOR )[0];
}

/**
 * Get the domain name of the first in the server chain (the rating author)
 */
function ratingGetAuthor()
{
	var uri = this.getAuthorURI();
	if ( !uri )
		return "";
	if ( gObjURIHome && gObjURIHome.spec == uri )
		return "Me";

	var uriParts = uri.match( '^http://(www.)?([^?]+)/*/' );
	if ( uriParts.length >= 3 )
		return uriParts[2];
	else
		return uri;
}
```

11.1.9 Review.xul

This XML file describes the user interface of the review dialog.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<dialog id="dsaOptionsDialog" title="Don't Shop Alone - Options"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  buttons="accept,cancel"
  ondialogaccept="return onAccept();"
  ondialogcancel="return onCancel();">

<script type="application/x-javascript" src="constants.js" />
<script type="application/x-javascript" src="util.js" />
<script type="application/x-javascript" src="Rating.js" />
<script type="application/x-javascript" src="io.js" />

<vbox>
      <label control="txtReview" value="Please explain briefly why you have rated this site:" />
      <textbox id="txtReview" maxlength="255" multiline="true" rows="3" />
      <description style='max-width: 400px'>
            Note that ratings are used to assess the reputation of the company behind the website
            (eg their customer service, product quality etc).
            Please do not assess the quality of the website itself.
      </description>
</vbox>

<script type="application/x-javascript">
<![CDATA[

var rating = window.arguments[0];
document.getElementById( "txtReview" ).value = rating.review;

function onAccept()
{
      // The rating is passed to this dialog as an argument
      rating.review = document.getElementById( "txtReview" ).value;

      if ( rating.review == "" && confirm( "No review entered. Would you like to retry?" ) )
            return false;
      else
            return true;
}

function onCancel()
{
      return true;
}

 ]]>
</script>


</dialog>
```

11.1.10 SiteState.js

This JavaScript file describes the object that holds the state of the currently-browsed website

```
/**
 * SiteState object encapsulates the state of a site being browsed
 */
function SiteState( argObjURI )
{
	this.objURI = argObjURI;
	this.objServerURI = null;
	this.ratings = null;
	this.initGetRatingsTimestamp = null;
	this.gotRatings = false;
	this.gotSniff = false;
	this.isUnprotectedServer = false;
	this.isSubscribed = false;
	this.reviewCount = 0;
	this.usersRating = null;
	this.isDSAServer = siteIsServer;
	this.setRatings = setRatings;
	this.parseRatings = parseRatings;
	this.isMyHome = siteIsMyHome;
	this.isDisplayable = siteIsDisplayable;
}

/**
 * Site is displayable if it has received the sniff and got ratings (if
 * there is a home server available
 */
function siteIsDisplayable()
{
	return ( this.isMyHome() ||
		( !gObjURIHome || this.gotRatings ) && this.gotSniff );
}

function siteIsServer()
{
	return ( this.objServerURI != null );
}

function siteIsMyHome()
{
	if ( !this.objServerURI || !gObjURIHome )
		return false
	return this.objServerURI.spec == gObjURIHome.spec;
}

function setRatings( argRatings )
{
	this.ratings = this.parseRatings( argRatings );
	this.gotRatings = true;
}

// Process ratings, setting other state variables where necessary
function parseRatings( ratings )
{
	if ( !ratings )
		return false;

	this.reviewCount = 0;
	this.isSubscribed = false;

	for( var i = 0; i < ratings.length; i++ )
	{
		// Check for user's vote amongst the ratings
		if ( gObjURIHome &&
			ratings[i].serverChain.indexOf( gObjURIHome.spec ) == 0 )
		{
			// User has submitted this rating
			this.usersRating = ratings[i];
```

```
                }
                if ( ratings[i].review )
                        this.reviewCount++;
        }
        // Check if the rating indicates a subscribed DSA server
        this.isSubscribed = ( this.usersRating &&
                this.usersRating.ratingValue == RATING_POSITIVE &&
                this.usersRating.isDSAServer );

        return ratings;
}
```

11.1.11 takeOwnershipOfServer.xul

This XML file describes the dialog presented when the user attempts to set the current protected server as their home server.

```
<?xml version="1.0"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>

<dialog id="dsaProtectServerDialog" title="Don't Shop Alone - Protect Server"
  xmlns="http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul"
  buttons="accept,cancel"
  ondialogaccept="return onAccept();"
  ondialogcancel="return onCancel();">

<script type="application/x-javascript">
<![CDATA[

function onAccept()
{
      // The rating is passed to this dialog as an argument
      var password = window.arguments[0];
      password.value = document.getElementById( "txtPassword" ).value;

      if ( !password.value )
      {
            alert( "Please enter a password" );
            return false;
      }
      else
            return true;
}

function onCancel()
{
      return true;
}

 ]]>
</script>

<vbox>
      <label control="txtReview" value="Please enter the password for your DontShopAlone server:"
/>
      <textbox id="txtPassword" maxlength="25" />
      <description style='max-width: 400px'>
            This is the password you chose when first installing this server
      </description>
</vbox>

</dialog>
```

11.1.12 util.js

This JavaScript file contains utility functions used elsewhere in the system.

```
var parity;

function getParity()
{
	if ( parity = 'odd' )
		parity = 'even';
	else
		parity = 'odd';

	return parity;
}

/**
 * Returns the URI currently being browsed
 */
function getCurrentURI()
{
	return window._content.document.location;
}

/**
 * Open the given URI in the browser
 */
function navigateToURI( URI )
{
	window._content.document.location.href = URI;
}

/**
 * Character encodes a string to be compatible with query string format
 */
function URLEncode( str )
{
	// Additional 'replace' calls are for URL-encoding quotes and double quotes
    return escape( str ).replace(/\+/g, '%2C').replace(/\"/g,'%22').replace(/\'/g, '%27');
}

/**
 * Regular expression taken from http://4umi.com/web/regex/domainfromurl.htm
 */
/* function getDomain( URI )
{
	var strURI = new String( URI );
	if ( !strURI || strURI == "" ) return null;
	var domain = strURI.match( ".?://(www\.)?([^/:]+)" );
	domain = ( domain && domain[ 2 ] ) ? domain[ 2 ] : '';
	return domain;
} */

/**
 * Take a DOM element and recursively output its elements to the console
 */
function dumpRecursively( objElement, index, level )
{
	var dumpText = ""
	for ( var i = 0; i < level; i++ )
	{
		dumpText += "   ";
	}
	if( !objElement ) return false;

	dumpText += "Element " + index + ": [" + objElement.tagName + "]";
	if( objElement.nodeValue)
	{
		dumpText += " = [" + objElement.nodeValue + "]";
	}
	dump( dumpText + "\n" );
```

```
	if ( !objElement.childNodes ) return;

	// Recurse into the element's children
	for ( var i = 0; i < objElement.childNodes.length; i++ )
	{
		dumpRecursively( objElement.childNodes[i], i, level + 1 );
	}
}

/**
 * A listener object that detects browser location changes and calls
 * the pageLoadHandler method
 */
var dsaListener =
{
	QueryInterface: function(aIID)
	{
		if (aIID.equals(Components.interfaces.nsIWebProgressListener) ||
		   aIID.equals(Components.interfaces.nsISupportsWeakReference) ||
		   aIID.equals(Components.interfaces.nsISupports))
			return this;
		throw Components.results.NS_NOINTERFACE;
	},
	onStateChange : function(aProgress,aRequest,aFlag,aStatus) {return 0;},
	onLocationChange: function( aProgress, aRequest, aURI )
	{
		pageLoadHandler( aURI );
		return 0;
	},
	onProgressChange : function(a,b,c,d,e,f){},
	onStatusChange : function(a,b,c,d){},
	onSecurityChange : function(a,b,c){},
	onLinkIconAvailable : function(a){}
}

function truncateURIFile( objURI )
{
	// Strip file / query string / named anchor from the path to get subpath
	var reGetSubpath = new RegExp( '.*/' )
	var subpath = objURI.path.match( reGetSubpath )[0];
	return gIOService.newURI( 'http://' + objURI.host + subpath, null, null)
}

function truncateURIPathAndFile( objURI )
{
	// Strip file / query string / named anchor from the path to get subpath
	return gIOService.newURI( 'http://' + objURI.host + "/", null, null)
}

function hasPath( objURI )
{
	var reGetSubpath = new RegExp( '.*/' )
	var subpath = objURI.path.match( reGetSubpath )[0];
	return ( subpath != "" && subpath != "/" );
}

function getPrefs()
{
	dump( "Getting DSA preferences\n" );
	try
	{
		var prefRoot = Components.classes["@mozilla.org/preferences-service;1"].
		  getService(Components.interfaces.nsIPrefService);
		return prefRoot.getBranch( DSA_PREFS_ROOT );
	}
	catch( e )
	{
		dump( "Exception getting main preferences branch " + e );
		return null;
	}
}
```

11.2 DSA Server

11.2.1 dsa.php

This file contains all the functionality of the DSA server. It has been deliberately combined into a single file in order to simplify the installation of the DSA server.

```
<?php

// Define constants
define( 'FILENAME', $_SERVER[ 'DOCUMENT_ROOT' ] . '/dsa.rdf' );
define( 'FILENAME_PREFS', $_SERVER[ 'DOCUMENT_ROOT' ] . '/dsaPrefs.php' );
define( 'DSA_VERSION', '0.5.0' );
// Number of levels of recursion when querying servers for ratings
define( 'MAXIMUM_RECURSION_LEVELS', 5 );
// Mappings for query string arguments
define( 'DOMAIN', 'domain' );
define( 'URI', 'uri' );
define( 'RATING_VALUE', 'ratingValue' );
define( 'REVIEW', 'review' );
define( 'SERVER_CHAIN', 'serverChain' );
define( 'REQUEST_CHAIN', 'requestChain' );
define( 'PASSWORD', 'password' );
define( 'LIST_RATINGS', 'list' );
define( 'IS_DSA_SERVER', 'isDSAServer' );
define( 'RATING_POSITIVE', 1 );
define( 'RATING_NEGATIVE', -1 );
define( 'WSDL', 'wsdl' );
define( 'SNIFF', 'sniff' );
define( 'QS_LIST', '?list=true' );
define( 'DSA_FILE', 'dsa.php' );
define( 'SEPARATOR', '&' );
define( 'DOMAIN_SEPARATOR', ';' );
// Web Service Namespace
define( 'WS_NS', "urn:RatingService" );

require_once( "nusoap.php" );

// Enable verbose error reporting
error_reporting( E_ALL );

// Error handler function
function errorHandler( $errno, $errstr, $errfile, $errline )
{
      $message = $errno . ": " . $errstr . "\n in " . $errfile . " [" .
            $errline . "] (" . getScriptPath() . ")\n\n";
      error_log( $message, 1, "chris@chrisbeach.co.uk" );
      error_log( $message, 3, $_SERVER[ 'DOCUMENT_ROOT' ] . "/phperror.log" );
      //die();
}

// set to the user defined error handler
$oldErrorHandler = set_error_handler( "errorHandler" );

// Choose a response based on the request
if ( isset( $_GET[ SNIFF ] ) )
{
      // Respond to a server 'sniff'
      header( "Content-Type: text/xml" );
      echo ( xmlWrap( null ) );
}
/* else if ( isset( $_POST[ RATING_VALUE ] ) || isset( $_GET[ RATING_VALUE ] ) )
{
      // Save a rating to this server
      header( "Content-Type: text/xml" );
      echo ( xmlWrap( saveRating() ) );
} */
else if ( isset( $_GET[ LIST_RATINGS ] ) )
{
```

```
	// List the ratings in HTML format
	echo getRatingsHTML();
}
else
{
	// Run the Web Service server
	webservice();
}

class Rating
{
	var $domain;
	var $uri;
	var $ratingValue;
	var $review = null;
	var $serverChain = "";
	var $isDSAServer = false;

	function Rating( $data )
	{
		if ( !$data )
		{
			trigger_error( "Invalid array passed to Rating constructor", E_USER_ERROR );
		}
		if ( is_array( $data ) )
		{
			// Data is already in the form of an array
			$dataArray = $data;
		}
		else
		{
			// Deserialise data from string to array
			$dataArray = array();
			$params = explode( SEPARATOR, $data );
			foreach( $params as $param )
			{
				$paramSplit = explode( "=", $param);
				if ( isset( $paramSplit[ 1 ] ) )
					$dataArray[ $paramSplit[ 0 ] ] = $paramSplit[ 1 ];
			}
		}
		if ( isset ( $dataArray[ DOMAIN ] ) )
			$this->domain = $dataArray[ DOMAIN ];
		if ( isset ( $dataArray[ URI ] ) )
			$this->uri = $dataArray[ URI ];
		if ( isset ( $dataArray[ RATING_VALUE ] ) )
			$this->ratingValue = $dataArray[ RATING_VALUE ];
		if ( isset ( $dataArray[ REVIEW ] ) )
			$this->review = $dataArray[ REVIEW ];
		if ( isset ( $dataArray[ SERVER_CHAIN ] ) )
			$this->serverChain = $dataArray[ SERVER_CHAIN ];
		if ( isset ( $dataArray[ IS_DSA_SERVER ] ) )
		{
			if ( $dataArray[ IS_DSA_SERVER ] == 'true' )
				$this->isDSAServer = true;
			else
				$this->isDSAServer = false;
		}
	}

	function toString()
	{
		$str = '';
		$str .= DOMAIN . '=' . $this->domain . SEPARATOR;
		$str .= URI . '=' . $this->uri . SEPARATOR;
		$str .= RATING_VALUE . '=' . $this->ratingValue . SEPARATOR;
		$str .= REVIEW . '=' . $this->review . SEPARATOR;
		$str .= SERVER_CHAIN . '=' . $this->serverChain . SEPARATOR;
		$str .= IS_DSA_SERVER . '=' . ( $this->isDSAServer ? 'true' : 'false' );
		return $str;
	}
```

```
	function getHeaderHTML()
	{
		$html = '';
		$html .= "<table>\n";
		$html .= "	<tr>\n";
		$html .= "		<th>Domain</th>\n";
		$html .= "		<th>Rating</th>\n";
		$html .= "		<th>Review</th>\n";
		$html .= "	</tr>\n";
		return $html;
	}

	function getFooterHTML()
	{
		$html = '';
		$html .= "</table>\n";
		return $html;
	}

	function getHTML()
	{
		// Separate domains with a comma
		$cssClass = ( $this->ratingValue == RATING_POSITIVE ?
			'positive' : 'negative' );
		$ratingValue = ( $this->ratingValue == RATING_POSITIVE ?
			'positive' : 'negative' );
		$html = '';
		$html .= "	<tr class='$cssClass'>\n";
		$html .= "		<td>{$this->domain}</td>\n";
		$html .= "		<td>$ratingValue</td>\n";
		$html .= "		<td>{$this->review}</td>\n";
		$html .= "	</tr>\n";
		return $html;
	}

	function getNameIndexedArray()
	{
		return array(
			DOMAIN => $this->domain,
			URI => $this->uri,
			RATING_VALUE => $this->ratingValue,
			REVIEW => $this->review,
			SERVER_CHAIN => $this->serverChain,
			IS_DSA_SERVER => $this->isDSAServer );
	}
}

class Server
{
	var $uri;

	function Server( $uri )
	{
		$this->uri = $uri;
	}
	function getHeaderHTML()
	{
		$html = '';
		$html .= "<table>\n";
		$html .= "	<tr>\n";
		$html .= "		<th>URI</th>\n";
		$html .= "		<th></th>\n";
		$html .= "	</tr>\n";
		return $html;
	}

	function getFooterHTML()
	{
		$html = '';
		$html .= "</table>\n";
		return $html;
	}
```

```
	function getHTML()
	{
		$listURI = $this->uri . DSA_FILE . QS_LIST;

		$html = '';
		$html .= "	<tr class='dsaServer'>\n";
		$html .= "		<td><a href='{$this->uri}'>{$this->uri}</a></td>\n";
		$html .= "		<td><a href='$listURI'>[list ratings]</a></td>\n";
		$html .= "	</tr>\n";
		return $html;
	}
}

function getSubscribedDSAServers()
{
	$ratings = getRatingsFromFile();
	$servers = array();
	foreach( $ratings as $rating )
	{
		if ( $rating->isDSAServer && $rating->ratingValue == RATING_POSITIVE )
			$servers[] = $rating;
	}
	return $servers;
}

function getRatings( $domain, $requestChain = '' )
{
	if ( isset( $requestChain[ '!xsi:nil' ] ) && $requestChain[ '!xsi:nil' ] == 'true' )
		$requestChainArray = array();
	else
		$requestChainArray = explode( DOMAIN_SEPARATOR, $requestChain );

	$soapRatings = array();
	foreach( getAggregatedRatings( $domain, $requestChainArray ) as $rating )
	{
		if ( $rating->serverChain )
			$rating->serverChain .= DOMAIN_SEPARATOR;
		$rating->serverChain .= getScriptPath();
		$soapRatings[] = $rating->getNameIndexedArray();
	}
	return $soapRatings;
}

function getRatingsFromDSAServers( $domain, $requestChainArray )
{
	$ratings = array();

	// Put a maximum limit on the level of recursion
	if ( count( $requestChainArray ) >= MAXIMUM_RECURSION_LEVELS )
		return $ratings;

	//trigger_error( "getRatingsFromDSAServers( [" . $domain . "], [ " .
	//	arrayStr( $requestChainArray ) . "]" );

	$dsaServers = getSubscribedDSAServers();

	foreach( $dsaServers as $dsaServer )
	{
		foreach( $requestChainArray as $alreadyVisitedServer )
		{
			if ( !$alreadyVisitedServer )
				continue( 1 );
			if ( stripURIScheme( $dsaServer->uri ) ==
				stripURIScheme( $alreadyVisitedServer ) )
			{
				// Don't aggregate the same server twice
				// (prevents circular loops)
				continue( 2 );
			}
			/* else
			{
```

```
                                trigger_error( "unmatched [ " . $alreadyVisitedServer .
                                        "] and [" . $dsaServer->uri . "]" );
                        } */
                }
                $newRequestChainArray = $requestChainArray;
                $newRequestChainArray[] = getScriptPath();

                $wsdl = $dsaServer->uri . "dsa.php?" . WSDL;
                $client = new soapclient( $wsdl, WSDL );
                $param = array(
                        DOMAIN => $domain,
                        REQUEST_CHAIN => implode( DOMAIN_SEPARATOR, $newRequestChainArray )
                );
                $soapRatings = $client->call( 'getRatings', $param );
                if ( $err = $client->getError() )
                {
                        trigger_error( "WS Error detected: " . $err );
                }
                if ( !$soapRatings ) continue;
                foreach( $soapRatings as $soapRating )
                {
                        $ratings[] = new Rating( $soapRating );
                }
        }
        return $ratings;
}

function getAggregatedRatings( $domain, $requestChainArray )
{
        $ratings = getRatingsFromFile( $domain );
        $ratings = array_merge( $ratings, getRatingsFromDSAServers( $domain, $requestChainArray ) );
        return $ratings;
}

function getRatingsFromFile( $domain = '' )
{
        $ratings = array();
        if ( !file_exists( FILENAME ) )
                return $ratings;

        // Get ratings on this server
        $fileString = file_get_contents( FILENAME );

        $ratingArray = explode( "\n", $fileString );
        foreach ( $ratingArray as $ratingString )
        {
                if ( !$ratingString )
                        continue;

                $rating = new Rating( $ratingString );

                if ( !$domain || $rating->domain == $domain )
                        $ratings[] = $rating;
        }
        return $ratings;
}

/**
 * Add a rating, or if ratingValue = 0, delete the rating
 */
function addRating( $rating, $password )
{
        $newRating = new Rating( $rating );

        if ( stripURIScheme( $newRating->uri ) == stripURIScheme( getScriptPath() ) )
                return new soap_fault( 'Client','','You cannot rate your own server' );

        if ( !isPasswordCorrect( $password ) )
                return new soap_fault( 'Client','','Password incorrect' );

        $oldRatingsArray = getRatingsFromFile();
        $newRatingsArray = array();
```

```
	foreach( $oldRatingsArray as $rating )
	{
		// Purge any ratings with the same domain as the new rating
		if ( $rating->domain != $newRating->domain )
			$newRatingsArray[] = $rating;
	}
	if ( $newRating->ratingValue != 0 )
		$newRatingsArray[] = $newRating;
	$fileString = '';

	// Concatenate string output from ratings objects
	foreach( $newRatingsArray as $rating )
	{
		$fileString .= $rating->toString() . "\n";
	}
	$fp = fopen( FILENAME, 'w' );
	if( !$fp )
		return new soap_fault( 'Client','','Could not open file' );

	$bytesWritten = fputs( $fp, $fileString );
	fclose( $fp );

	if( $bytesWritten == 0 )
		return new soap_fault( 'Client','','Data failed to save' );

	return true;
}

function changePassword( $oldPassword, $newPassword )
{
	if( !isPasswordCorrect( $oldPassword ) )
		return new soap_fault( 'Client','','Could not change password - old password
incorrect' );

	addPref( PASSWORD, $newPassword );
	return true;
}

/**
 * Checks whether the password is correct for this server
 * - If preference file exists, check against the password stored there
 * - If not, save the given password and return true
 */
function isPasswordCorrect( $password )
{
	$prefs = getPrefs();
	if ( isset( $prefs[ PASSWORD ] ) )
	{
		return ( $prefs[ PASSWORD ] == $password );
	}
	else
	{
		addPref( PASSWORD, $password );
		return true;
	}
}

/**
 * Serve the webservices
 */
function webservice()
{
	$httpRawPostData = file_get_contents( 'php://input' );
	//trigger_error( "webservice() HTTP_RAW_POST_DATA=[$httpRawPostData]" );

	$server = new soap_server();
	$server->configureWSDL( 'RatingsService', WS_NS );
	$server->wsdl->schemaTargetNamespace = WS_NS;
	$server->wsdl->addComplexType(
		'Rating', 'complexType', 'struct', 'all', '',
		array(
			DOMAIN => array( 'name' => DOMAIN, 'type' => 'xsd:string' ),
```

```
                        URI => array( 'name' => URI, 'type' => 'xsd:string' ),
                        RATING_VALUE => array( 'name' => RATING_VALUE, 'type' => 'xsd:integer' ),
                        REVIEW => array( 'name' => REVIEW, 'type' => 'xsd:string' ),
                        SERVER_CHAIN => array( 'name' => SERVER_CHAIN, 'type' => 'xsd:string' ),
                        IS_DSA_SERVER => array( 'isDSAServer' => IS_DSA_SERVER, 'type' =>
'xsd:boolean' )
                )
        );

        $server->wsdl->addComplexType(
                'RatingArray', 'complexType', 'array', '', 'SOAP-ENC:Array', array(),
                array( array( 'ref'=>'SOAP-ENC:arrayType', 'wsdl:arrayType'=>'tns:Rating[]' ) ),
'tns:Rating' );

        $server->register( 'addRating', array(
                        'rating' => 'tns:Rating',
                        'password' => 'xsd:string' ),
                array( 'return' => 'xsd:boolean' ), WS_NS
        );

        $server->register( 'getRatings', array(
                        'domain' => 'xsd:string',
                        'requestChain' => 'xsd:string' ),
                array( 'return' => 'tns:RatingArray' ), WS_NS
        );

        $server->register( 'changePassword',
                array(
                        'oldPassword' => 'xsd:string',
                        'newPassword' => 'xsd:string' ),
                array( 'return' => 'tns:RatingArray' ), WS_NS
        );

        $server->service( $httpRawPostData );
}

function getRatingsHTML()
{
        $ratingsAndServers = getRatingsFromFile();
        $html = '';

        $html .= "<html>\n";
        $html .= "<head><title>Ratings</title>\n";
        $html .= "      <style type='text/css'>\n";
        $html .= "              body { font-family: verdana }\n";
        $html .= "              h1, h2, body { color: #333 }\n";
        $html .= "              h1 { font-size: 14pt }\n";
        $html .= "              h2 { font-size: 12pt }\n";
        $html .= "              table { border-collapse: collapse; border: 1px solid #333 }\n";
        $html .= "              tr { text-align: left }\n";
        $html .= "              th { background-color: #333; color: #fff; font-weight: normal }\n";
        $html .= "              td, th { padding: 0 6px 0 3px; font-size: 10pt }\n";
        $html .= "              td { border-left: 1px solid #999; border-bottom: 1px solid #999 }\n";
        $html .= "              tr.positive td { background-color: #cec; color: #242 }\n";
        $html .= "              tr.negative td { background-color: #ecc; color: #422 }\n";
        $html .= "              tr.dsaServer td { background-color: #cce; color: #224 }\n";
        $html .= "      </style>\n";
        $html .= "</head>\n";
        $html .= "<body>\n";

        $servers = array();
        $ratings = array();

        $html .= "<h1>DontShopAlone Server - " . stripURIScheme( getScriptPath() ) . "</h1>\n";

        foreach( $ratingsAndServers as $ratingOrServer )
        {
                if ( $ratingOrServer->isDSAServer )
                        $servers[] = new Server( $ratingOrServer->uri );
                else
                        $ratings[] = $ratingOrServer;
        }
```

```
	if ( $ratings )
	{
		$html .= "<h2>Ratings:</h2>\n";
		$html .= Rating::getHeaderHTML();
		foreach( $ratings as $rating )
		{
			$html .= $rating->getHTML();
		}
		$html .= Rating::getFooterHTML();
	}
	else
		$html .= "<h2>No Ratings Yet</h2>";
	if ( $servers )
	{
		$html .= "<h2>Subscribed Servers:</h2>\n";
		$html .= Server::getHeaderHTML();
		foreach( $servers as $server )
		{
			$html .= $server->getHTML();
		}
		$html .= Server::getFooterHTML();
	}
	$html .= "</body>\n";
	$html .= "</html>\n";
	return $html;
}

function xmlWrap( $argXML )
{
	$prefs = getPrefs();
	$isProtected = isset( $prefs[ PASSWORD ] ) ? 'true' : 'false';

	$xml = '';

	$xml .= "<?xml version='1.0' encoding='UTF-8' standalone='yes'?>\n";
	$xml .= "<dsaresponse version='" . DSA_VERSION . "'";
	$xml .=		" isprotected='" . $isProtected . "'>\n";
	$xml .= $argXML . "\n";
	$xml .= "</dsaresponse>\n";

	return $xml;
}

/**
 * Add the given preference to the preferences file
 */
function addPref( $newKey, $newValue )
{
	$prefs = getPrefs();
	$prefs[ $newKey ] = $newValue;

	// Create the content for the file
	$fileString = '';
	$fileString .= "<?php\n";
	$fileString .= "/* DontShopAlone Preferences\n";
	$fileString .= " * Please do not modify this file\n";
	foreach( $prefs as $key => $value )
	{
		$fileString .= "$key=$value\n";
	}
	$fileString .= " */\n";
	$fileString .= "?>";

	// Open and write the file
	$fp = fopen( FILENAME_PREFS, 'w' );
	$bytesWritten = fputs( $fp, $fileString );
}

/**
 * Get a name-indexed array of preferences
 */
function getPrefs()
```

```
{
	$prefs = array();
	if ( file_exists( FILENAME_PREFS ) )
	{
		$fileString = file_get_contents( FILENAME_PREFS );
		// Parse the file, looking for the password setting
		$filePrefs = explode( "\n", $fileString );
		foreach ( $filePrefs as $pref )
		{
			$prefParts = explode( "=", $pref );
			if ( isset( $prefParts[1] ) )
				$prefs[ $prefParts[0] ] = $prefParts[1];
		}
	}
	return $prefs;
}

function getScriptPath()
{
	if ( ereg ( '.*/', $_ENV[ 'SCRIPT_URI' ], $regs ) )
	{
		return $regs[0];
	}
	else
		return $_ENV[ 'SCRIPT_URI' ];
}

function truncate( $str, $maxLength )
{
	$trailing = ' ...';

	if ( strlen( $str . $trailing ) < $maxLength )
		return $str;
	else
		return substr( $str, 0, $maxLength - $trailing ) . $trailing;
}

function arrayStr( $arr )
{
	$str = '';
	if ( !is_array( $arr ) )
		return $arr;
	foreach( $arr as $key=>$val )
	{
		if(is_array($val))
		{
			$str .= "[$key = {". arrayStr($val)."}] , ";
		}
		else
		{
			$str .= "[$key=$val] , ";
		}
	}
	return substr($str, 0, strlen($str)-3);
}

function getParity()
{
	global $parity;
	if ( $parity == "odd" )
		$parity = "even";
	else
		$parity = "odd";
	return $parity;
}

function stripURIScheme( $argURI )
{
	$uri = str_replace( "http://www.", "", $argURI );
	return str_replace( "http://", "", $uri );
}
```

```
?>
```